به نام خدا

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

درس بازیابی هوشمند اطلاعات

پاسخ بخش تئوری تمرین ۳

نام و نام خانودگی: روژین پناو

شماره دانشجویی: ۲۲۰۷۰۱۰۴۶

مهر ماه ۱۴۰۳

تأیید می‌کنم که ازLLM ها مطابق با دستورالعمل‌های بارگذاری شده در سامانه Elearn درس به طور مسئولانه استفاده کرده‌ام، تمام اجزای کار خود را درک می‌کنم و آماده بحث شفاهی درباره آن‌ها هستم.

فهرست

پاسخ سوال اول

پاسخ بخش الف

در این سوال، مسئله‌ای که پیش اومده اینه که وقتی داده‌ها زیاد می‌شن، حافظه پر میشه و باعث میشه پردازش‌ها خیلی کند بشن. برای حل این مشکل، می‌تونیم از **compression** و **sort-based methods** استفاده کنیم.

```
class Mapper:

    procedure Map(docid n, doc d):

        H ← new AssociativeArray

        for all term t ∈ doc d do

            H[t] ← H[t] + 1

        for all term t ∈ H do

            Emit(term t, posting <n, H[t]>)


class Reducer:

    procedure Reduce(term t, postings <[n1, f1], [n2, f2] ...])

        P ← new List

        for all posting <a, f> ∈ postings do

            Append(P, <a, f>)

        Sort(P)

        Emit(term t, postings P)
```

در کد اولیه، داده‌ها در **Mapper** به صورت (term, posting) به مرحله‌ی **Reduce** ارسال می‌شوند. در **Reducer**، تمامی پستینگ‌ها برای هر **term** در حافظه ذخیره می‌شوند و سپس مرتب‌سازی می‌شوند. در صورتی که تعداد **term** ها و **posting** ها زیاد باشد، این مرحله می‌تواند به سرعت باعث پر شدن حافظه شود و در نهایت با **scalability bottleneck** مواجه شویم.

راه حل:

در الگوریتم اولیه، داده‌ها به صورت **(term, posting)** در حافظه ذخیره می‌شوند و به مرحله‌ی **Reducer** ارسال می‌شوند. اما وقتی داده‌ها زیاد می‌شوند، حافظه پر می‌شود و سرعت پردازش کند می‌شود. برای جلوگیری از این مشکل، باید داده‌ها را به صورت **key-value pair** ذخیره و پردازش

کنیم. این به این معناست که برای هر **term**، یک **key** و برای هر **docid**، یک **value** (که شامل فرکانس آن **term** در داکیومنت است) ذخیره می‌کنیم.

$$(compressed\_docid, \{term_i : frequency_i\})$$

در ادامه برای کار با مجموعه بسیار بزرگ از روش‌هایی مثل **compression** و **sort-based methods** میتوانیم استفاده کنیم تا از مصرف بیش از حد حافظه جلوگیری کنیم و داده‌ها را به شیوه‌ای بهینه پردازش کنیم.

:Sort-based Methods

برای اینکه داده‌ها در مرحله‌ی **Reduce** به صورت مرتب پردازش شوند، معمولاً نیاز به **sort** داریم. در کد اولیه، داده‌ها باید در **Reducer** مرتب شوند که این می‌تواند باعث مصرف زیاد حافظه و کاهش کارایی شود. با استفاده از **sort-based methods** می‌توانیم این مشکل را حل کنیم.

در روش **sort-based,** داده‌ها به جای اینکه در **Reducer** مرتب شوند، به صورت تدریجی و در مراحل قبلی مرتب می‌شوند. در مرحله‌ی **Mapper** داده‌ها به صورت **key-value pair** پردازش و ذخیره می‌شوند. سپس در مرحله‌ی **Shuffle**، داده‌ها به صورت محلی بر اساس **term** مرتب می‌شوند و در نهایت در **Reducer** تنها داده‌های مرتب‌شده بر اساس **docid** پردازش می‌شوند.

باعث:

- **مرتب‌سازی پیش ازReduce** : به جای اینکه همه داده‌ها در **Reducer** مرتب شوند، داده‌ها از قبل در **Shuffle** مرتب می‌شوند. این باعث می‌شود که **Reducer** فقط داده‌های مرتب‌شده را دریافت کند و نیازی به مرتب‌سازی مجدد نداشته باشد.
- **صرفه‌جویی در حافظه** : چون داده‌ها از قبل مرتب شده‌اند و دیگر نیازی به مرتب‌سازی در حافظه نیست، حجم حافظه کاهش می‌یابد.
- **حافظه بهینه‌تر** : مرتب‌سازی داده‌ها در مراحل قبلی باعث می‌شود که حافظه به صورت بهینه‌تری استفاده شود و از **scalability** **bottleneck**جلوگیری می‌شود.

:Compression

در Sort-based Methods یکی از مشکلات اصلی این است که داده‌ها به صورت **(docid, {term: frequency})** به **Reducer** ارسال می‌شوند، که می‌تواند حجم زیادی از داده‌ها را در حافظه ایجاد کند. برای کاهش این حجم، از **compression** می‌توانیم استفاده کنیم. به عنوان مثال، می‌توانیم از **Unary compression** برای فشرده‌سازی شناسه‌های داکیومنت (**docid**) استفاده کنیم. در این روش به جای ذخیره‌سازی خود **docid**، یک مقدار فشرده‌تر مثل باقی‌مانده تقسیم بر ۱۰۰۰ را ذخیره می‌کنیم که حجم داده‌ها را کاهش می‌دهد.

باعث:

- **کاهش حجم داده‌ها:** با فشرده‌سازی داده‌ها، مقدار حافظه مورد نیاز برای ذخیره‌سازی داده‌ها در هر مرحله کاهش می‌یابد.
- **افزایش سرعت پردازش :** چون داده‌ها کوچکتر می‌شوند، سرعت پردازش و انتقال داده‌ها سریع‌تر می‌شود و از فشار اضافی بر حافظه جلوگیری می‌شود.

پاسخ بخش ب

در مرحله اول ، برای هر docid داده‌ها به صورت

(compressed_docid, {term: frequency})

محاسبه می‌شوند و داده‌ها بر اساس docid مرتب میشوند.

```
class Mapper:
    def map(self, docid, doc):
        term_frequency = {}

        for term in doc:
            term_frequency[term] = term_frequency.get(term, 0) + 1
        compressed_docid = self.compress(docid)

        self.emit(compressed_docid, term_frequency)

    def compress(self, docid):
        return docid % 1000
```

در این مرحله، داده‌ها از فایل موقت خوانده می‌شوند و به صورت **خارج از حافظه** بر اساس term مرتب می‌شوند.

```
class LocalSorter:
    def local_sort(self, postings_file_path: str) -> list:
        sorted_postings = []

        with open(postings_file_path, 'r') as f:
            for line in f:
```

```
                term, freq = line.strip().split(":")
                sorted_postings.append((term, int(freq)))

        sorted_postings.sort(key=lambda x: x[0])

        return sorted_postings
```

در این مرحله، داده‌های مرتب شده از چندین **Mapper** با هم ترکیب می‌شوند. از **heap** برای ادغام داده‌ها استفاده می‌شود.

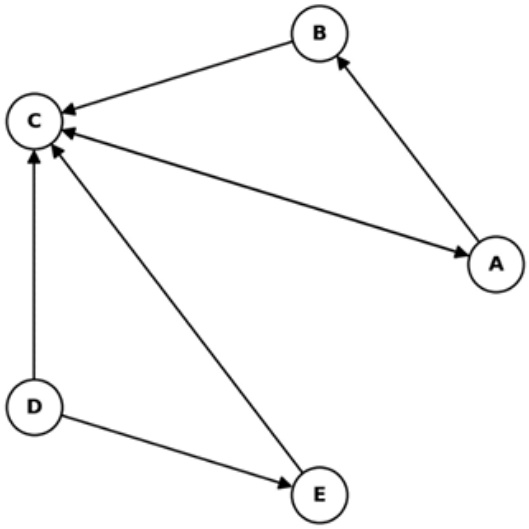
```
class MergeSorter:
    def merge_sorted_data(self, sorted_data_list):
        merged_data = []

        heap = []
        for data in sorted_data_list:
            heapq.heappush(heap, data)

        while heap:
            smallest_term = heapq.heappop(heap)
            merged_data.append(smallest_term)

        return merged_data
```

در این مرحله، داده‌ها به صورت مرتب‌شده بر اساس docid وارد **Reducer** می‌شوند و یک **Inverted Index** نهایی تولید می‌شود.

```
class Reducer:
    def reduce(self, term, postings):
        sorted_postings = sorted(postings, key=lambda x: x[0])
        self.emit(term, sorted_postings)
```

پـاسخ سوال دوم

پـاسخ بـخش الـف



TRANSITION MATRIX = M

	A	B	C	D	E
A	0	1	0	0	0
B	0	0	1	0	0
C	0	0	0	0	0
D	0	0	1/2	0	1/2
E	0	0	1	0	0

$$p(d_j) = \sum_{i=1}^{N} \left[\frac{1}{N}\alpha + (1-\alpha)M_{ij}\right] p(d_i)$$

$$\vec{p} = (\alpha I + (1-\alpha)M)^T \vec{p}$$

1

$$A = (1-\alpha)M + \alpha I = 0.85 \times \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} + 0.15 \times \begin{bmatrix} 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \end{bmatrix}$$

تکرار اول , $\alpha = 0.15$

$$\begin{bmatrix} P^1(A) \\ P^1(B) \\ P^1(C) \\ P^1(D) \\ P^1(E) \end{bmatrix} = a^T \times \begin{bmatrix} P^0(A) \\ P^0(B) \\ P^0(C) \\ P^0(D) \\ P^0(E) \end{bmatrix} = \begin{bmatrix} 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.88 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.3 & 0.88 & 0.3 & 0.455 & 0.3 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.3 & 0.3 & 0.3 & 0.455 & 0.3 \end{bmatrix} \times \begin{bmatrix} 1/5 \\ 1/5 \\ 1/5 \\ 1/5 \\ 1/5 \end{bmatrix} = \begin{bmatrix} 0.3 \\ 0.416 \\ 0.447 \\ 0.3 \\ 0.331 \end{bmatrix}$$

تکرار دوم , $\alpha = 0.15$

$$\begin{bmatrix} P^2(A) \\ P^2(B) \\ P^2(C) \\ P^2(D) \\ P^2(E) \end{bmatrix} = a^T \times \begin{bmatrix} P^1(A) \\ P^1(B) \\ P^1(C) \\ P^1(D) \\ P^1(E) \end{bmatrix} = \begin{bmatrix} 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.88 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.3 & 0.88 & 0.3 & 0.455 & 0.3 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.3 & 0.3 & 0.3 & 0.455 & 0.3 \end{bmatrix} \times \begin{bmatrix} 0.3 \\ 0.416 \\ 0.447 \\ 0.3 \\ 0.331 \end{bmatrix} = \begin{bmatrix} 0.54 \\ 0.71 \\ 0.83 \\ 0.54 \\ 0.58 \end{bmatrix}$$

پاسخ بخش ب

$$A = (1-\alpha)M + \alpha I = 1 \times \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} + 0 \times \begin{bmatrix} 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \end{bmatrix}$$

تکرار اول , $\alpha = 0$

$$\begin{bmatrix} P^1(A) \\ P^1(B) \\ P^1(C) \\ P^1(D) \\ P^1(E) \end{bmatrix} = a^T \times \begin{bmatrix} P^0(A) \\ P^0(B) \\ P^0(C) \\ P^0(D) \\ P^0(E) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 \end{bmatrix} \times \begin{bmatrix} 1/5 \\ 1/5 \\ 1/5 \\ 1/5 \\ 1/5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.2 \\ 0.3 \\ 0 \\ 0.1 \end{bmatrix}$$

تـکـرار دوم $\alpha = 0,$

$$\begin{bmatrix} P^2(A) \\ P^2(B) \\ P^2(C) \\ P^2(D) \\ P^2(E) \end{bmatrix} = a^T \times \begin{bmatrix} P^1(A) \\ P^1(B) \\ P^1(C) \\ P^1(D) \\ P^1(E) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0.2 \\ 0.3 \\ 0 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0.2 \\ 0 \\ 0 \end{bmatrix}$$

پـاسخ بـخش ج

ADJACENCY MATRIX = A

	A	B	C	D	E
A	0	1	0	0	0
B	0	0	1	0	0
C	0	0	0	0	0
D	0	0	1	0	1
E	0	0	1	0	0

$$\boldsymbol{h(d_i)} = \sum_{\boldsymbol{d_j \in \mathrm{OUT}(d_i)}} \boldsymbol{a(d_j)}$$

$$\boldsymbol{a(d_i)} = \sum_{\boldsymbol{d_j \in \mathrm{IN}(d_i)}} \boldsymbol{h(d_j)}$$

تـکـرار اول

$$\begin{bmatrix} \boldsymbol{a}(A) \\ \boldsymbol{a}(B) \\ \boldsymbol{a}(C) \\ \boldsymbol{a}(D) \\ \boldsymbol{a}(E) \end{bmatrix} = \sum_{\boldsymbol{d_j \in \mathrm{IN}(d_i)}} \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 3 \\ 0 \\ 1 \end{bmatrix} \xrightarrow{normalize} \begin{bmatrix} 1/6 \\ 1/6 \\ 1/2 \\ 0 \\ 1/6 \end{bmatrix}$$

$$\begin{bmatrix}\boldsymbol{h}(A)\\ \boldsymbol{h}(B)\\ \boldsymbol{h}(C)\\ \boldsymbol{h}(D)\\ \boldsymbol{h}(E)\end{bmatrix} = \sum_{d_j \in \text{OUT}(d_i)} \begin{bmatrix}0 & 1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & 0\\ 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0 & 1\end{bmatrix} = \begin{bmatrix}1\\1\\1\\1\\2\end{bmatrix} \xrightarrow{normalize} \begin{bmatrix}1/6\\1/6\\1/6\\1/6\\1/3\end{bmatrix}$$

تکرار دوم

$$\begin{bmatrix}\boldsymbol{a}(A)\\ \boldsymbol{a}(B)\\ \boldsymbol{a}(C)\\ \boldsymbol{a}(D)\\ \boldsymbol{a}(E)\end{bmatrix} = \sum_{d_j \in \text{IN}(d_i)} \begin{bmatrix}0 & 0 & 1/6 & 0 & 0\\ 1/6 & 0 & 0 & 0 & 0\\ 1/6 & 1/6 & 0 & 1/6 & 1/3\\ 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1/3\end{bmatrix} = \begin{bmatrix}1/6\\1/6\\5/6\\0\\1/3\end{bmatrix} \xrightarrow{normalize} \begin{bmatrix}1/9\\1/9\\5/9\\0\\2/9\end{bmatrix}$$

$$\begin{bmatrix}\boldsymbol{h}(A)\\ \boldsymbol{h}(B)\\ \boldsymbol{h}(C)\\ \boldsymbol{h}(D)\\ \boldsymbol{h}(E)\end{bmatrix} = \sum_{d_j \in \text{OUT}(d_i)} \begin{bmatrix}0 & 1/6 & 0 & 0 & 0\\ 0 & 0 & 1/2 & 0 & 0\\ 1/6 & 0 & 0 & 0 & 0\\ 0 & 0 & 1/2 & 0 & 0\\ 0 & 0 & 1/2 & 0 & 1/6\end{bmatrix} = \begin{bmatrix}1/6\\1/2\\1/6\\1/2\\4/6\end{bmatrix} \xrightarrow{normalize} \begin{bmatrix}1/12\\1/4\\1/12\\1/4\\1/3\end{bmatrix}$$

پاسخ بخش د

نتایج حالت A = 0.15

- بعد از دو تکرار، همه گره‌ها از جمله D و E مقدار غیر صفر داشتن :
  - D ≈ 0.54
  - E ≈ 0.58
- وجود **DAMPING FACTOR** باعث شد که احتمال‌ها به‌صورت تصادفی در کل گراف پخش بشن .(TELEPORTATION) این ویژگی جلوی گیر افتادن در مؤلفه های بسته رو گرفت و به گره های حاشیه ای مثل D و E هم سهمی داد.

نتایج حالت A = 0

- بعد از دو تکرار، مقدار **PAGERANK** برای گره‌های D و E برابر صفر شد.

- دلیلش اینه که وقتی **DAMPING FACTOR** وجود نداره، احتمال‌ها فقط بر اساس لینک‌ها حرکت می‌کنن. در این حالت، چون گراف دارای مسیرهای گیرنده (SINK) هست، رتبه‌ها به سمت گره C جذب میشن و اونجا باقی می‌مونن.
- نتیجه: گره‌های ضعیف‌تر مثل D و E کاملاً حذف میشن و هیچ سهمی از رتبه نمی‌گیرن.

پس، **DAMPING FACTOR** مثل یک نجات‌دهنده برای گره‌های ضعیف عمل می‌کنه. بدونش، گره‌هایی مثل D و E هیچ شانسی برای دیده شدن ندارن و کل رتبه به سمت گره‌های قوی‌تر میره. با وجودش، حتی گره‌های حاشیه‌ای هم سهمی پیدا می‌کنن و الگوریتم به یک حالت پایدار و منصفانه می‌رسه.

پاسخ بخش ه

با توجه به خروجی‌ها، در بخش الف الگوریتم **PAGERANK** مقادیر نهایی این‌طورند: A=0.54 ، B=0.71، C=0.83، D=0.54، E=0.58؛

و در بخش ج الگوریتم **HITS** برای **AUTHORITY** داریم:

A=1/9 ، B=1/9، C=5/9، D=0، E=2/9.

گره **C** هم بالاترین PAGERANK را دارد (۰,۸۳) و هم بالاترین AUTHORITY را در HITS(5/9). بنابراین گره برتر در هر دو معیار یکی است.

این هم‌خوانی از نظر مفهومی هم منطقی است. در الگوریتم HITS ، گره C چون توسط چند هاب معتبر لینک شده، بیشترین اعتبار یا AUTHORITY را کسب می‌کند. از طرف دیگر در PAGERANK ، جریان احتمال در شبکه به سمت گره‌هایی می‌رود که بیشترین لینک ورودی را دارند و در این گراف، C مقصد اصلی بسیاری از لینک‌هاست. علاوه بر این، وجود **DAMPING FACTOR** باعث می‌شود که حتی در صورت وجود مسیرهای بسته، احتمال‌ها دوباره به سمت گره‌های مهم مثل C برگردند.

در واقع، این نتیجه نشان می‌دهد که گره C نه‌تنها از دیدگاه محلی (LOCAL) و رابطه‌ی مستقیم با هاب‌ها در HITS مهم است، بلکه از دیدگاه جهانی (GLOBAL) و توزیع احتمال در کل شبکه در PAGERANK هم اهمیت بالایی دارد. پس به این نتیجه میتوانیم برسیم که C در ساختار این گراف نقش کلیدی و مرکزی دارد و طبیعی است که در هر دو روش رتبه‌ی بالاتری نسبت به سایر گره‌ها بگیرد.

پـاسخ سـوال سـوم

ورودی اول :

$$\begin{bmatrix}(A)\\(B)\\(C)\\(D)\\(E)\end{bmatrix} = \begin{bmatrix}0.54\\0.71\\0.83\\0.54\\0.58\end{bmatrix}$$

ورودی دوم :

A	search retrieval HITS
B	web search graph
C	HITS algorithm graph retrieval
D	web link analysis
E	link graph search

: MapReduce Pseudocode

Map Function

تـولـید (word, (docID, position)) بـرای همه کـلمـات

```
MAP(docID, text):
    words = text.split()
    for position, word in enumerate(words):
        EMIT(word, (docID,  [position]))
```

Shuffle & Sort

- MapReduce بطور خودکـار خروجی Map را بـر اسـاس key (کـلمه) گـروه‌بـنـدی مـی‌کـنـد.
- پـس از گـروه‌بـنـدی، بـرای هر کـلمه داریـم لـیست (docID, [positions]).

Reduce Function

```
REDUCE(word, list_of_doc_positions):
    doc_positions_with_pagerank = []

    for (docID, positions) in list_of_doc_positions:
        pagerank = lookup_pagerank(docID)
        doc_positions_with_pagerank.append((docID, pagerank, positions))

    sorted_docs = sort(doc_positions_with_pagerank, key=pagerank, descending=True)
    EMIT(word, sorted_docs)
```

گام ۱: ساخت Positional Index

ابتدا موقعیت کلمات در هر سند را مشخص می‌کنیم:

Doc	متن	Positional Index
A	search retrieval HITS	search:[0], retrieval:[1], HITS:[2]
B	web search graph	web:[0], search:[1], graph:[2]
C	HITS algorithm graph retrieval	HITS:[0], algorithm:[1], graph:[2], retrieval:[3]
D	web link analysis	web:[0], link:[1], analysis:[2]
E	link graph search	link:[0], graph:[1], search:[2]

گام ۲ : map output

<search, (A,[0])>

<retrieval, (A,[1])>

<HITS, (A,[2])>

<web, (B,[0])>

<search, (B,[1])>

<graph, (B,[2])>

<HITS, (C,[0])>

<algorithm, (C,[1])>

<graph, (C,[2])>

<retrieval, (C,[3])>

<web, (D,[0])>

<link, (D,[1])>

<analysis, (D,[2])>

<link, (E,[0])>

<graph, (E,[1])>

<analysis, (E,[2])>

گام ۳: INTERNAL GROUPING

<algorithm, (C, 0.83,[1])>

<algorithm, (D, 0.54,[2])>

<algorithm, (E, 0.58,[2])>

<graph, (B, 0.71,[2])>

<graph, (C, 0.83,[2])>

<graph, (E, 0.58,[1])>

<HITS, (A, 0.54,[2])>

<HITS, (C, 0.83,[0])>

<link, (D, 0.54,[1])>

<link, (E, 0.58,[0])>

<retrieval, (A, 0.54,[1])>

<retrieval, (C, 0.83,[3])>

<search, (A, 0.54,[0])>

<search, (B, 0.71,[1])>

<web, (B, 0.71,[0])>

<web, (D, 0.54,[0])>

گام 4: REDUCE OUTPUT

<algorithm, [(C, 0.83,[1]), (E, 0.58,[2]), (D, 0.54,[2])]>

<graph, [(C, 0.83,[2]), (B, 0.71,[2]), (E, 0.58,[1])]>

<HITS, [(C, 0.83,[0]), (A, 0.54,[2])]>

<link, [(E, 0.58,[0]), (D, 0.54,[1])]>

<retrieval, [(C, 0.83,[3]), (A, 0.54,[1])]>

<search, [(B, 0.71,[1]), (A, 0.54,[0])]>

<web, [(B, 0.71,[0]), (D, 0.54,[0])]>