Quiz — 1 — Answer

MIR — Spring 2026

Instructor: Dr. Amin Eskandari

Head Teaching Assistants: Hamid Namjoo, Amir Hossein Hemati

Teaching Assistants: Ali Mojahed, Ali Nikvan, Amir Javvi

پاسخ سوال 1

بخش الف:

کوئری داده شده :

(A OR B) AND (C OR D)

برای حل این نوع کوئری در مدل بولی باید مراحل زیر را انجام دهیم :

مرحله ۱ محاسبه (A OR B)

A → (1, 3, 4, 8, 10)

B → (1, 2, 4, 8)

عملگر OR به معنی اجتماع (Union) دو لیست است .بنابراین باید همه شناسه اسنادی که در A یا Bوجود دارند را جمع کنیم

نتیجه :

A OR B → (1, 2, 3, 4, 8, 10)

مرحله ۲ محاسبه (C OR D)

C → (2, 4, 6, 8, 10)

D → (1, 4, 7, 8)

اجتماع دو لیست :

C OR D → (1, 2, 4, 6, 7, 8, 10)

مرحله ۳ –محاسبه ANDبین دو نتیجه

اکنون باید اشتراک دو مجموعه زیر را پیدا کنیم :

(1, 2, 3, 4, 8, 10)

(1, 2, 4, 6, 7, 8, 10)

عملگر AND یعنی اشتراک (Intersection)

اسناد مشترک :

(1, 2, 4, 8, 10)

نتیجه نهایی بخش الف

اسناد بازیابی‌شده :

1، 2 ، 4 ، 8 ، 10

بخش ب

کوئری اصلی :

(A OR B) AND (C OR D)

کوئری بازنویسی‌شده توسط دانشجو :

(A AND C) OR (A AND D) OR (B AND C) OR (B AND D)

این بازنویسی در واقع از قانون توزیعی در منطق بولی استفاده می‌کند.

قانون توزیعی :

(X OR Y) AND (Z OR W)

معادل است با :

(X AND Z) OR (X AND W) OR (Y AND Z) OR (Y AND W)

بنابراین از نظر منطق بولی این دو عبارت کاملاً معادل هستند .

پس ادعای دانشجو کاملا درست است و هر دو کوئری همیشه دقیقاً همان مجموعه اسناد را تولید می‌کنند .

نکته مهم:

معادل بودن از نظر نتیجه منطقی به این معنی نیست که از نظر هزینه اجرا نیز یکسان باشند .

بخش ج:

اندازه Posting Listها :

A → 1,000,000

B → 800,000

C → 10,000

D → 5,000

کوئری :

(A OR B) AND (C OR D)

گام ۱ بررسی پیشنهاد دانشجو

دانشجو می‌گوید:

همیشه باید کوچک‌ترین لیست‌ها را اول intersect کنیم

کوچک‌ترین لیست‌ها :

D = 5K

C = 10K

اما مشکل اینجاست که در این پرس‌وجو ابتدا ORوجود دارد نه AND

یعنی :

(A OR B)

(C OR D)

بنابراین نمی‌توان مستقیم C و Dرا intersectکرد .

گام ۲ –بررسی اندازه نتایج میانی

(A OR B)

از آنجا که هر دو لیست بسیار بزرگ هستند :

A = 1M

B = 800K

نتیجه OR تقریباً نزدیک به 1.8میلیون سند خواهد بود .

(C OR D)

C = 10K

D = 5K

نتیجه ORتقریباً :

حدود 15Kسند

گام ۳ –اجرای AND

در مرحله آخر :

1.8M AND 15K

این عملیات نسبتا ارزان است زیرا می‌توان با الگوریتم merge سریع انجام داد .

نتیجه تحلیلی

قانون کوچک‌ترین لیست اول فقط برای زنجیره ANDها کاملاً معتبر است .

مثال :

A AND B AND C AND D

در چنین حالتی بهتر است :

کوچک‌ترین لیست‌ها ابتدا intersect شوند .

اما دراین سؤال به دلیل وجود OR ساختار پرس‌وجو متفاوت است و این قانون همیشه بهترین استراتژی نیست .

نتیجه :

ادعای دانشجو همیشه درست نیست .

بخش د

آیا با Boolean Index می‌توان رتبه‌بندی انجام داد؟

فرض شده است :

ایندکس فقط Boolean است .

یعنی فقط این اطلاعات ذخیره شده :

Term → List of Documents

و مشخص می‌کند که :

آیا واژه در سند وجود دارد یا نه .

اطلاعاتی که در Boolean Index وجود ندارد

برای رتبه‌بندی در مدل‌هایی مانند TF-IDF معمولاً به این اطلاعات نیاز داریم :

1. Term Frequency (TF)

تعداد تکرار واژه در هر سند

2. Document Length

طول سند

3. Term Weight

اما در Boolean Index هیچ‌کدام از این اطلاعات ذخیره نشده‌اند .

نتیجه مهم

در چنین ایندکسی فقط می‌توان فهمید:

آیا واژه در سند وجود دارد یا نه.

اما نمی‌توان فهمید:

کدام سند مرتبط‌تر است .

آیا می‌توان رتبه‌بندی انجام داد؟

دو حالت وجود دارد .

حالت اول رتبه‌بندی واقعی TF-IDF

غیرممکن است، زیرا مقدار TF در دسترس نیست .

حالت دوم رتبه‌بندی ساده

می‌توان یک روش ساده استفاده کرد :

تعداد واژه‌های کوئری که در سند وجود دارند.

مثال:

Query = information retrieval models

اگر سندی هر سه واژه را داشته باشد → امتیاز بیشتر

اگر فقط دو واژه را داشته باشد → امتیاز کمتر

اما این روش بسیار ساده است و کیفیت آن بسیار پایین‌تر از TF-IDF است .

پاسخ سوال 2

ساخت ماتریس وقوع بولی و فهرست معکوس

ابتدا باید بررسی کنیم هر واژه در کدام سند ظاهر شده است .

اسناد:

D1: web search engines use ranking models

D2: boolean retrieval model for web search

D3: information retrieval models and ranking techniques

D4: web mining techniques for search systems

D5: boolean query processing in retrieval systems

D6: ranking models for information systems

واژگان مورد بررسی:

web

search

retrieval

ranking

boolean

models

مرحله ۱ –بررسی حضور هر واژه در اسناد

واژه web

در اسناد زیر وجود دارد :

D1

D2

D4

واژه search

در اسناد زیر وجود دارد:

D1

D2

D4

واژه retrieval

در اسناد زیر وجود دارد:

D2

D3

D5

واژه ranking

در اسناد زیر وجود دارد:

D1

D3

D6

واژه boolean

در اسناد زیر وجود دارد:

D2

D5

واژه models

در اسناد زیر وجود دارد:

D1

D3

D6

مرحله ۲ ساخت ماتریس وقوع بولی

در ماتریس وقوع:

اگر واژه در سند وجود داشته باشد 1

اگر وجود نداشته باشد 0

ترتیب ستون‌ها:

D1 D2 D3 D4 D5 D6

web

1 1 0 1 0 0

search

1 1 0 1 0 0

retrieval

0 1 1 0 1 0

ranking

1 0 1 0 0 1

boolean

0 1 0 0 1 0

models

1 0 1 0 0 1

مرحله ۳ ساخت فهرست معکوس

در فهرست معکوس به جای ماتریس، برای هر واژه لیستی از اسناد نگهداری می‌شود .

web → (D1, D2, D4)

search → (D1, D2, D4)

retrieval → (D2, D3, D5)

ranking → (D1, D3, D6)

boolean → (D2, D5)

models → (D1, D3, D6)

نکته:

ماتریس وقوع از نظر مفهومی ساده است اما از نظر حافظه بسیار پرهزینه است.

فهرست معکوس بسیار فشرده‌تر و در موتورهای جستجو واقعی استفاده می‌شود .

بخش ب : ارزیابی پرس‌وجوی بولی

کوئری:

(web AND search) OR (retrieval AND ranking) OR boolean

مرحله ۱ محاسبه (web AND search)

web → (D1, D2, D4)

search → (D1, D2, D4)

اشتراک دو لیست و نتیجه:

(D1, D2, D4)

مرحله ۲ محاسبه (retrieval AND ranking)

retrieval → (D2, D3, D5)

ranking → (D1, D3, D6)

اشتراک:

D3

نتیجه:

(D3)

مرحله ۳ لیست boolean

boolean → (D2, D5)

مرحله ۴ OR بین نتایج

(D1, D2, D4)

(D3)

(D2, D5)

اجتماع:

(D1, D2, D3, D4, D5)

نتیجه نهایی

اسناد بازیابی شده:

D1, D2, D3, D4, D5

بخش ج : بررسی معادل بودن دو پرس‌وجو

کوئری اصلی:

(web AND search) OR (retrieval AND ranking) OR boolean

پرس‌وجوی پیشنهادی:

(web OR retrieval OR boolean)

AND

(search OR ranking OR boolean)

مرحله ۱ تحلیل منطقی

در پرس‌وجوی دوم اگر سندی فقط شامل واژه boolean باشد چه می‌شود؟

عبارت اول:

(web OR retrieval OR boolean)

درست است .

عبارت دوم:

(search OR ranking OR boolean)

همچنان درست است .

پس هر سندی که فقط boolean داشته باشد نیز بازیابی می‌شود .

اما در پرس‌وجوی اول:

boolean به صورت مستقل OR شده است

بنابراین اسناد دارای boolean نیز بازیابی می‌شوند.

اما ترکیب شرط‌ها در پرس‌وجوی دوم می‌تواند اسناد بیشتری را شامل شود .

مثال

فرض کنید سندی فقط این واژه‌ها را دارد:

retrieval

search

در کوئری اول:

(retrieval AND ranking)<<<غلط

(web AND search)<<<غلط

booleanندارد

پس سند بازیابی نمی‌شود .

اما در کوئری دوم :

(retrieval OR ...)<<<درست

(search OR ...)<<<درست

پس سند بازیابی می‌شود .

نتیجه:

دو کوئری معادل نیستند .

بخش د : بهینه سازی ترتیب پردازش

اندازه posting listها:

web → 40M

search → 30M

retrieval → 500K

ranking → 200K

boolean → 20K

models → 5M

کوئری:

(web AND search AND models)

OR

(retrieval AND ranking AND boolean)

اصل مهم

در عملیات AND باید کوچک‌ترین لیست‌ها ابتدا intersect شوند .

قسمت دوم پرس‌وجو

retrieval → 500K

ranking → 200K

boolean → 20K

ترتیب بهینه:

boolean AND ranking

نتیجه حدودا 20K ≤

سپس

(result) AND retrieval

قسمت اول کوئری

web → 40M

search → 30M

models → 5M

ترتیب بهینه:

models AND search

سپس

(result) AND web

چرا این ترتیب بهتر است؟

اگر ابتدا دو لیست بزرگ intersectشوند:

web AND search

عملیات بسیار پرهزینه خواهد شد.

ولی اگر ابتدا لیست کوچک‌تر استفاده شود:

اندازه نتایج میانی بسیار کوچک‌تر می‌شود .

بخش ه : مقایسه مدل‌های بازیابی

Query

information retrieval models ranking

مدل بولی ساده

در این مدل:

یا سند کاملا ًشرط را دارد یا ندارد.

مثلاً:

retrieval AND models AND ranking

اسنادی که هر سه واژه را دارند بازیابی می‌شوند.

در مجموعه اسناد:

D3این سه واژه را دارد .

مشکل مدل بولی

دو سند ممکن است هر دو شرط را داشته باشند اما:

یکی بسیار مرتبط‌تر باشد.

مدل بولی این تفاوت را تشخیص نمی‌دهد .

Extended Boolean Model

در این مدل به جای 0 و 1 از درجه شباهت استفاده می‌شود.

مثلاً:

سندی که 3 واژه از 4 واژه query را دارد امتیاز بیشتری می‌گیرد .

Ranked Retrieval

در این مدل برای هر سند یک score محاسبه می‌شود.

معمولاً با روش‌هایی مثل:

TF-IDF

BM25

در این مدل:

اسناد بر اساس میزان ارتباط مرتب می‌شوند .

بخش و : طراحی رتبه‌بندی با Boolean Index

فرض کنید فقط این اطلاعات موجود است:

Term → list of documents

و هیچ اطلاعات فرکانسی نداریم .

روش پیشنهادی

می‌توان از روش زیر استفاده کرد:

Query term counting

برای هر سند:

تعداد واژه‌های query که در سند وجود دارند شمارش می‌شود.

مثال:

Query = information retrieval models ranking

اگر سند:

4واژه داشته باشد → score = 4

3واژه داشته باشد → score = 3

مثال

D3

information

retrieval

models

ranking

score = 4

محدودیت‌های این روش

۱) نمی‌توان فهمید واژه چند بار تکرار شده است .

۲) تفاوت اهمیت واژه‌ها مشخص نیست .

۳) طول سند در نظر گرفته نمی‌شود .

نتیجه نهایی

Boolean Indexبرای رتبه‌بندی بسیار محدود است.

به همین دلیل موتورهای جستجوی واقعی از مدل‌های رتبه‌بندی مانند :

TF-IDF

BM25

Neural Ranking

استفاده می‌کنند.

پاسخ سوال 3

بخش (الف) چالش سرعت: استفاده از Skip Pointers

هدف سؤال

پیدا کردن گزارش‌هایی که هر دو واژه زیر را دارند:

Sharingan AND Amaterasu

اندازه لیست‌ها:

Sharingan → 500,000 سند

Amaterasu → 50 سند

این یعنی یک لیست بسیار بزرگ و یک لیست بسیار کوچک داریم.

مرحله ۱ – یادآوری نحوه کار الگوریتم AND معمولی

در بازیابی بولی، برای محاسبه:

A AND B

از الگوریتم دو نشانگر استفاده می‌شود.

روش کار:

دو اشاره‌گر روی ابتدای دو لیست قرار می‌گیرند.

در هر مرحله:

1) اگر DocID ها برابر باشند

آن سند در پاسخ قرار می‌گیرد.

2) اگر DocID اول کوچک‌تر باشد

اشاره‌گر لیست اول جلو می‌رود.

3) اگر DocID دوم کوچک‌تر باشد

اشاره‌گر لیست دوم جلو می‌رود.

مرحله ۲ مشکل در این سؤال

اندازه لیست‌ها بسیار متفاوت است.

Sharingan → 500,000

Amaterasu → 50

بدون بهینه‌سازی چه اتفاقی می‌افتد؟

برای هر DocID در لیست Amaterasu باید در لیست بسیار بزرگ Sharingan حرکت کنیم تا به همان DocID برسیم.

در نتیجه تعداد زیادی مقایسه انجام می‌شود.این کار باعث می‌شود زمان اجرا زیاد شود.

مرحله ۳ — ایده Skip Pointers

Skip Pointer یعنی:

اضافه کردن اشاره‌گرهای اضافی که اجازه می‌دهند

به جای حرکت قدم‌به‌قدم،

چندین عنصر را یکجا رد کنیم.

مثال ساده

فرض کنید لیست Sharingan این‌گونه باشد:

1 → 3 → 8 → 15 → 21 → 35 → 60 → ...

اگر Skip Pointer داشته باشیم:

1 → 3 → 8 → 15 → 21 → 35 → 60

| | |

↓ ↓ ↓

15 35 80

یعنی از 1 می‌توان مستقیماً به 15 پرید.

مرحله ۴ نحوه ساخت Skip Pointer ها

قاعده رایج در طراحی Skip Pointer:

تعداد پرش‌ها تقریباً برابر است با:

n√

که n طول لیست است.

در اینجا:

n = 500000

تقریباً:

√500000 ≈ 707

پس:

حدود هر 700 عنصر یک Skip Pointer قرار می‌دهیم.

یعنی:

Doc1 → Doc701

Doc701 → Doc1401

Doc1401 → Doc2101

و به همین ترتیب.

مرحله ۵ — نحوه استفاده از Skip Pointer هنگام AND

فرض کنید می‌خواهیم DocID = 20000 از لیست Amaterasu را در لیست Sharingan پیدا کنیم.

بدون Skip Pointer:

باید از ابتدای لیست حرکت کنیم:

1 → 3 → 8 → 15 → ...

تا به 20000 برسیم.

این ممکن است هزاران مقایسه نیاز داشته باشد.

اما با Skip Pointer:

سیستم ابتدا بررسی می‌کند

آیا مقدار Skip هنوز از 20000 کوچک‌تر است؟

مثلاً:

skip → 701 → 1

skip → 1401 → 701

skip → 2101 → 1401

...

وقتی مقدار بعدی از 20000 بزرگ‌تر شد،

آنگاه حرکت عادی انجام می‌دهیم.

در نتیجه به جای هزاران حرکت،

فقط چند پرش بزرگ انجام می‌شود.

مرحله ۶ نتیجه استفاده از Skip Pointer

مزایا:

1) کاهش شدید تعداد مقایسه‌ها

2) سریع‌تر شدن AND بین لیست‌های نامتوازن

3) مناسب برای لیست‌های بسیار بزرگ

در این مثال:

Skip Pointer باعث می‌شود جستجو در لیست 500,000 تایی بسیار سریع‌تر انجام شود.

بخش (ب) چالش ذخیره‌سازی: انتخاب ساختار داده

در این بخش باید بین دو ساختار انتخاب کنیم:

Variable-length Array

Linked List

ابتدا ویژگی هرکدام را بررسی می‌کنیم.

ساختار اول Variable-length Array

این ساختار شبیه یک آرایه معمولی است.

ویژگی‌ها:

1) عناصر پشت سر هم در حافظه قرار دارند.

2) دسترسی بسیار سریع است.

3) جستجو و پیمایش سریع انجام می‌شود.

اما:

درج عنصر جدید سخت است.

چرا؟

چون ممکن است لازم باشد

کل آرایه جابه‌جا شود.

ساختار دوم Linked List

در این ساختار:

هر عنصر شامل دو بخش است:

DocID

Pointer به عنصر بعدی

مزایا:

1) درج عنصر بسیار آسان

2) نیاز به جابه‌جایی عناصر نیست

اما:

معایب:

1) دسترسی کندتر

2) داده‌ها در حافظه پراکنده هستند

3) استفاده از cache کمتر مؤثر است

تحلیل سؤال

دو نوع تکنیک داریم:

تکنیک قدیمی

لیست طولانی

تقریباً ثابت

تکنیک جدید

لیست کوتاه

مدام در حال تغییر

انتخاب برای تکنیک قدیمی

برای لیست طولانی و ثابت بهترین انتخاب:

Variable-length Array

دلیل:

1) جستجو سریع‌تر

2) پیمایش سریع‌تر

3) درج جدید تقریباً انجام نمی‌شود

پس مشکل درج اهمیت ندارد.

انتخاب برای تکنیک جدید

برای لیست کوتاه و در حال تغییر:

بهترین انتخاب:

Linked List

دلیل:

1) درج بسیار سریع

2) نیاز به جابه‌جایی عناصر نیست

3) تغییرات مکرر راحت انجام می‌شود

نتیجه نهایی

تکنیک قدیمی با لیست بلند

Variable-length Array

تکنیک جدید با لیست کوتاه و متغیر

Linked List

این انتخاب تعادل بین:

سرعت جستجو و سهولت درج

را برقرار می‌کند.

بخش (پ) چالش مکان‌مندی: Positional Index

هدف سؤال

پیدا کردن اسنادی که در آن‌ها:

Naruto

Sasuke

در فاصله حداکثر 2 کلمه از هم قرار دارند.

مرحله ۱ ساختار Inverted Index معمولی

در ساده‌ترین حالت:

برای هر واژه فقط DocID ذخیره می‌شود.

مثال:

Naruto → (D1, D3, D5)

Sasuke → (D2, D3, D5)

از این اطلاعات فقط می‌دانیم:

این واژه در چه سندی آمده است.

اما نمی‌دانیم:

در کجای سند آمده است.

مرحله ۲ مشکل Inverted Index معمولی

فرض کنید:

Naruto → (D3)

Sasuke → (D3)

می‌دانیم هر دو در سند D3 هستند.

اما نمی‌دانیم:

آیا کنار هم هستند یا نه.

ممکن است:

Naruto در ابتدای سند باشد

Sasuke در انتهای سند باشد.

در این حالت فاصله آن‌ها بسیار زیاد است.

پس Inverted Index ساده نمی‌تواند

فاصله واژه‌ها را بررسی کند.

مرحله ۳ راه حل: Positional Index

در Positional Index علاوه بر DocID،

موقعیت کلمه در سند نیز ذخیره می‌شود.

مثال:

Naruto

D3: (5, 20, 70)

Sasuke

D3: (7, 21, 80)

یعنی:

Naruto در موقعیت‌های 5 و 20 و 70 آمده است.

مرحله ۴ بررسی فاصله واژه‌ها

حال می‌توان فاصله را بررسی کرد.

مثال:

Naruto = 5

Sasuke = 7

فاصله:

2 = 5 - 7

پس شرط فاصله حداکثر 2 برقرار است.

در نتیجه این سند معتبر است.

مرحله ۵ نتیجه

Inverted Index معمولی فقط می‌تواند بگوید:

یک واژه در چه سندی وجود دارد.

اما نمی‌تواند بگوید:

واژه‌ها در چه فاصله‌ای از هم هستند.

برای پاسخ به پرسش‌هایی مانند:

عبارت دقیق

یا فاصله بین واژه‌ها

باید از:

Positional Index

استفاده کنیم.

پاسخ سوال 4

بخش (الف) نوشتن بولی کوئری صورت مسئله

ما باید بازیکنانی را پیدا کنیم که:

- یا تانک باشند

- یا هیلر باشند

- اما سرعت پایین نداشته باشند

در بازیابی اطلاعات، برای چنین شرایطی از مدل بولی استفاده می‌کنیم.

در مدل بولی سه عملگر اصلی داریم:

AND >>> هر دو شرط باید برقرار باشد

OR >>> حداقل یکی از شرط‌ها برقرار باشد

NOT >>> حذف یک ویژگی

مرحله ۱ تشخیص شرایط مثبت

بازیکنان مورد نظر باید یکی از این دو ویژگی را داشته باشند:

تانک یا هیلر

پس می‌نویسیم:

Tank OR Healer

مرحله ۲ تشخیص شرط منفی

بازیکنان نباید ویژگی زیر را داشته باشند:

LowSpeed

پس باید این ویژگی را حذف کنیم:

NOT LowSpeed

مرحله ۳ – ترکیب شروط

ما می‌خواهیم:

کسانی که تانک یا هیلر هستند

و در عین حال سرعت پایین ندارند.

پس از AND استفاده می‌کنیم.

کوئری نهایی:

(Tank OR Healer) AND NOT LowSpeed

سیستم ابتدا بازیکنانی را پیدا می‌کند که:

Tank OR Healer

سپس از میان آن‌ها کسانی که:

LowSpeed

هستند حذف می‌شوند.

در نتیجه فقط بازیکنان مناسب باقی می‌مانند.

بخش (ب) بهینه‌سازی کوئری (Query Optimization)

اطلاعات مسئله

لیست تانک‌ها:

5000 بازیکن

لیست هیلرها:

100 بازیکن

اصل مهم در بهینه‌سازی کوئری بولی

در سیستم‌های بازیابی اطلاعات، برای عملیات AND یا پردازش لیست‌ها معمولاً از این قانون استفاده می‌شود:

ابتدا کوچک‌ترین لیست پردازش می‌شود.

دلیل این کار:

کاهش تعداد مقایسه‌ها.

تحلیل مسئله

اندازه دو لیست:

Tank = 5000

Healer = 100

واضح است که:

لیست هیلرها بسیار کوچک‌تر است.

چرا باید از لیست کوچک‌تر شروع کنیم؟

اگر از لیست کوچک‌تر شروع کنیم:

فقط 100 عنصر بررسی می‌شود.

اما اگر از لیست بزرگ‌تر شروع کنیم:

باید 5000 عنصر بررسی شود.

در سیستم‌هایی که میلیون‌ها داده دارند، این تفاوت بسیار مهم است.

نتیجه

سیستم باید ابتدا لیست:

Healer

را بررسی کند.

دلیل:

کوچک‌تر بودن لیست و کاهش هزینه محاسباتی.

بخش (پ) الگوریتم Intersection (اشتراک)

هدف

پیدا کردن بازیکنانی که:

Tank AND RangedDamage

هستند.

در این حالت باید اشتراک دو لیست را پیدا کنیم.

فرض کنید لیست‌ها به صورت زیر باشند

لیست Tank

40 , 20 , 15 , 8 , 2

لیست RangedDamage

50 , 25 , 20 , 8 , 3

مرحله ۱ قرار دادن دو اشاره‌گر

یک اشاره‌گر روی ابتدای هر لیست قرار می‌دهیم.

Tank pointer → 2

Ranged pointer → 3

مرحله ۲ مقایسه

3 > 2

پس اشاره‌گر لیست کوچکتر جلو می‌رود.

Tank pointer → 8

مرحله ۳ مقایسه جدید

8 = 8

پس این یک عضو مشترک است.

نتیجه:

8

هر دو اشاره‌گر جلو می‌روند.

Tank → 15

Ranged → 20

مرحله ۴ مقایسه

20 > 15

پس اشاره‌گر Tank جلو می‌رود.

Tank → 20

مرحله ۵ مقایسه

20 = 20

این هم یک عضو مشترک است.

نتیجه:

20

هر دو اشاره‌گر جلو می‌روند.

Tank → 40

Ranged → 25

مرحله ۶ ادامه مقایسه

40 > 25

پس اشاره‌گر Ranged جلو می‌رود.

Ranged → 50

حالا:

50 > 40

پس Tank جلو می‌رود.

Tank → پایان لیست

الگوریتم متوقف می‌شود.

نتیجه نهایی اشتراک

بازیکنانی که هم تانک و هم دمیج‌زن راه دور هستند:

20 , 8

بخش (ج) تأثیر هزینه عملگر NOT

صورت سؤال

اگر هزینه پردازش عملگر NOT بیشتر از OR باشد،

آیا باز هم قانون «شروع از کوچک‌ترین لیست» برقرار است؟

ابتدا باید مفهوم هزینه پردازش را بفهمیم.

در بازیابی اطلاعات، هر عملگر هزینه‌ای دارد:

AND → مقایسه بین دو لیست

OR → ادغام دو لیست

NOT → حذف عناصر از یک مجموعه

عملگر NOT معمولاً گران‌تر است چون:

باید یک مجموعه بزرگ از اسناد بررسی شود.

مثال

در دیتابیس 1,000,000 بازیکن

اگر بگوییم:

NOT LowSpeed

سیستم باید بررسی کند:

کدام بازیکنان سرعت پایین ندارند.

این کار ممکن است نیاز به بررسی تعداد زیادی سند داشته باشد.

آیا قانون کوچک‌ترین لیست همچنان برقرار است؟

پاسخ: بله، اما با یک نکته مهم.

قاعده شروع از کوچک‌ترین لیست هنوز مفید است

زیرا تعداد مقایسه‌ها را کاهش می‌دهد.

اما وقتی عملگر NOT در کوئری وجود دارد،

سیستم‌های واقعی معمولاً سعی می‌کنند:

ابتدا بخش‌های مثبت کوئری را محاسبه کنند

و سپس NOT را اعمال کنند.

مثال

در کوئری:

(Tank OR Healer) AND NOT LowSpeed

سیستم ابتدا محاسبه می‌کند:

Tank OR Healer

سپس از نتیجه، بازیکنان LowSpeed حذف می‌شوند.

نتیجه نهایی

حتی اگر هزینه NOT بیشتر از OR باشد:

اصل شروع از کوچک‌ترین لیست همچنان مفید است،

اما موتور جستجو معمولاً ابتدا بخش‌های مثبت کوئری را پردازش می‌کند

و در مرحله آخر عملگر NOT را اعمال می‌کند.