Data Structures and algorithms
Instructor : **Dr.Amin Eskandari**
Head-Ta's : Hamid Namjoo manesh , Amir Hossein Hemati , Ali Mojahed
Ta's : Amir Mohammad Asadjoo , Padina razmjooi , Armin Janfaza , Alireza Heydari , Mahdi Torabi ,
Maryam Ghorbani , Fatemeh Keshavarz , MohammadReza Fasli , Abolfazl Tadrisi
Guest TA : Fatemeh Amiri

Quiz 05 Answers

پاسخنامه آزمون شماره ۰۵

بخش ۱: نوار جستجو(Trie Search)

برای شروع، ما یک ساختار Trie داریم که کتاب‌ها داخل آن ذخیره شده‌اند. ایده Trie این است که هر گره نشان‌دهنده یک حرف است و مسیر از ریشه تا یک گره، یک پیشوند از نام کتاب‌ها را می‌سازد. در هر گره، لیستی از کتاب‌هایی که از آن مسیر عبور کرده‌اند نگه‌داری می‌شود، تا بتوانیم سریع بفهمیم چه کتاب‌هایی با یک پیشوند شروع می‌شوند.

بخش (الف): جستجوی پیشوند

اول باید توجه کنیم که پیشوند جستجو شده برای هر دانشجو متفاوت است. این پیشوند یا "Da" است یا"The R" . فرض می‌کنیم حالت اول یعنی "Da" را داریم (اگر حالت شما "The R" باشد دقیقا همان منطق ولی روی مسیر دیگر اجرا می‌شود)

برای انجام جستجو، از ریشه Trie شروع می‌کنیم:

در ریشه هستیم و هیچ حرفی هنوز خوانده نشده است .

حرف اول پیشوند "D" است. بنابراین به فرزند 'D' در ریشه می‌رویم .

اگر این گره وجود داشته باشد، یعنی حداقل یک کتاب با حرف D شروع شده است، پس ادامه می‌دهیم .

حرف دوم "a" است، پس از گره D وارد فرزند 'a' می‌شویم .

وقتی به این گره رسیدیم، یعنی دقیقا به زیر درخت تمام کتاب‌هایی رسیده‌ایم که با "Da"شروع می‌شوند .

در این نقطه، دیگر نیازی به ادامه دادن مسیر نداریم، چون هدف ما فقط پیدا کردن پیشوند بود. حالا از داخل این گره، لیست کتاب‌ها را برمی‌داریم.

در این داده‌ها کتاب‌هایی که با "Da" شروع می‌شوند این‌ها هستند:

Dawn با 890 بار جستجو

Dark Matter با 410 بار جستجو

Dusk با 230 بار جستجو

Darkly با 125 بار جستجو

حالا باید این کتاب‌ها را بر اساس تعداد جستجو مرتب کنیم تا محبوب‌ترین‌ها اول باشند.

پس ترتیب نهایی:

1. Dawn (890)
2. Dark Matter (410)
3. Dusk (230)
4. Darkly (125)

پیچیدگی زمانی

برای تحلیل زمان، دو بخش داریم:

اول پیمایش Trie برای رسیدن به پیشوند. اگر طول پیشوند را p بگیریم، باید p مرحله حرکت کنیم. پس این بخش برابر است با:

p

بعد از آن، k کتاب پیدا کرده‌ایم. حالا باید آن‌ها را مرتب کنیم. اگر از Merge Sort استفاده کنیم، زمان آن می‌شود:

k log k

پس کل زمان:

p + k log k

بخش (ب): مقایسه دو کاربر

در این قسمت دو کاربر داریم:

- کاربر اول: "Dar" با 2 نتیجه
- کاربر دوم: "The R" با 3 نتیجه

حالا باید ببینیم آیا زمان اجرا برای این دو برابر است یا نه.

برای کاربر اول: طول پیشوند 3 است ، 2 نتیجه دارد، پس زمان : 3 + 2 log 2

برای کاربر دوم: طول پیشوند 5 است، 3 نتیجه دارد پس زمان : 5 + 3 log 3

بنابراین: کاربر دوم هم در پیمایش Trie بیشتر زمان می‌برد (چون مسیر طولانی‌تر است) و هم در مرتب‌سازی زمان بیشتری نیاز دارد(چون k بزرگ‌تر است) پس این دو عملیات از نظر زمانی برابر نیستند و کاربر دوم هزینه بیشتری دارد.

بخش (ج)Merge Sort :

درخت بازگشت Merge Sort و مراحل تقسیم و ادغام باید توسط دانشجو رسم شود (بسته به لیست خودش). فرض می‌کنیم لیست خروجی ما این است:

[890, 125, 410, 230]

Merge Sort به این شکل کار می‌کند که اول لیست را نصف می‌کند تا به تک عنصر برسد، سپس دوباره آن‌ها را با هم merge می‌کند.

مرحله اول تقسیم:

[410, 230] و [890, 125]

سپس دوباره:

[890], [125], [410], [230]

حالا مرحله ترکیب:

[890, 125] → [125, 890]

[410, 230] → [230, 410]

حالا دوباره ترکیب نهایی:

[125, 890] و [230, 410] → [125, 230, 410, 890]

پس درخت بازگشت به شکل تقسیم و merge مرحله‌ای کامل می‌شود.

در نهایت لیست مرتب‌شده بر اساس محبوبیت (نزولی) خواهد بود:

[890, 410, 230, 125]

بخش ۲: برترین‌های ماه

در این بخش هدف این است که فقط ۵ کتاب برتر را نگه داریم، نه اینکه کل داده‌ها را مرتب کنیم. چون تعداد کتاب‌ها بسیار زیاد است، مرتب‌سازی کامل هزینه زیادی دارد.

بخش (الف): انتخاب ساختار داده

Min-Heap با اندازه ثابت ۵ (برای نگهداری ۵ کتاب برتر)

در این Min-Heap، کوچک‌ترین امتیاز از میان ۵ کتاب برتر همیشه در ریشه قرار دارد. به این ترتیب، به راحتی می‌توانیم چک کنیم که آیا کتاب جدید ارزش وارد شدن به جمع ۵ تای برتر را دارد یا خیر. اگر امتیاز کتاب جدید از ریشه (کوچک‌ترین امتیاز فعلی) بیشتر باشد، آن را جایگزین می‌کنیم.

ساخت گام‌به‌گام Heap :

مرحله ۱: وارد کردن پنج کتاب اول کتاب‌های اولیه را به ترتیب ظاهر شدن در جدول وارد Heapمی‌کنیم:

Dune (4.8)

The Road (4.2)

Dawn (3.9)

Darkly (4.5)

The Rock (3.7)

پس از وارد کردن این پنج کتاب و انجام Heapify، وضعیت Heap:

ریشه :

3.7 (The Rock)

بقیه اعضا:

4.8 (Dune), 4.2 (The Road), 3.9 (Dawn), 4.5 (Darkly)

مرحله ۲: پردازش کتاب Dark Matter (4.1)

امتیاز ۴.۱ را با ریشه (۳.۷) مقایسه می‌کنیم.

چون ۴.۱ > ۳.۷ است، کتاب The Rock را حذف کرده و Dark Matter را وارد Heap می‌کنیم.

سپس Heapify انجام می‌شود تا خاصیت Min-Heap حفظ شود.

مراحل را برای باقی کتاب ها تکرار میکنیم.

۵ کتاب برتر نهایی (به ترتیب امتیاز نزولی):

1. Flowers for Algernon (4.9)
2. Dune (4.8)
3. 1984 (4.7)
4. Dusk (4.6)
5. Darkly (4.5)

بخش (ب): پیچیدگی زمانی

برای هر کتاب جدید فقط یک عملیات insert در Heap انجام می‌دهیم.

این عملیات برابر است با : log k

چون Heap فقط اندازه ۵ دارد، بسیار کوچک است. پس کل زمان: n log k این خیلی بهتر از مرتب‌سازی کامل n log n است.

بخش (ج): کتاب جدید

امتیاز کتاب جدید به صورت زیر حساب می‌شود:

$$S = (SumOfDigits(StudentID) \% 5) + 1$$

یعنی این مقدار بین 1 تا 5 خواهد بود. حالا بررسی می‌کنیم:

اگر S از کوچک‌ترین مقدار در Heap بزرگ‌تر باشد، وارد ۵ تای برتر می‌شود و جای کوچک‌ترین را می‌گیرد. اگر کوچک‌تر باشد، حذف می‌شود.

بخش ۳: شبکه اجتماعی(Graph)

بخش (الف): BFS ازNasim تا Armin

BFS یعنی جستجو لایه‌ای. از Nasim شروع می‌کنیم:

صف اولیه: [Nasim]

Nasim به Danial وصل است :

صف: [Danial]

Danial به Raha و Ryan وصل است:

صف: [Raha, Ryan] (چون ترتیب الفبایی رعایت می‌شود، Raha قبل از Ryan بررسی خواهد شد)

Raha به Mina و Armin وصل است:

صف: [Ryan, Mina, Armin]

در این مرحله Armin کشف می‌شود. بنابراین کوتاه‌ترین مسیر پیدا شده است.

کوتاه‌ترین زنجیره فالو: Nasim → Danial → Raha → Armin

بخش (ب): اشتراک پست‌ها

در این شبکه، اگر کاربر A ، کاربر B را فالو کند، یعنی پست‌های B به A نمایش داده می‌شود.
پس جهت انتشار پست، خلاف جهت یال‌های follow است.

برای پیدا کردن بینندگان یک پست، باید بررسی کنیم چه کاربرانی می‌توانند به صاحب پست برسند.

بینندگان پست Armin

کاربرانی که مستقیم یا غیرمستقیم به Armin می‌رسند :

Mina → Armin
Raha → Armin
Ryan → Mina → Armin
Danial → Raha → Armin
Sara → Danial → Raha → Armin

Nasim → Danial → Raha → Armin

پس مجموعه بینندگان پست Armin برابر است با:

{Mina, Raha, Ryan, Danial, Sara, Nasim}

بینندگان پستNasim

هیچ کاربری Nasim را فالو نمی‌کند و مسیری نیز به او وجود ندارد. پس هیچ‌کس پست Nasim را نمی‌بیند.

مجموعه بینندگان: ∅

اشتراک مجموعه بینندگان Armin و Nasim برابر است با: ∅

پس هیچ کاربری هر دو پست را مشاهده نمی‌کند. دلیل این موضوع این است که Nasimهیچfollower ی ندارد.

بخش (ج): viral بودن

برای اینکه یک پست به همه برسد، گراف باید strongly connected باشد. اما اینجا اینطور نیست چون Nasim هیچ ورودی ندارد. پس ادعا غلط است.

بخش ۴: مسیر خواندن(Topological Sort)

این بخش یک گراف وابستگی است.

بخش (الف): ترتیب مطالعه

ابتدا تمام کتاب‌هایی که in-degree آن‌ها برابر صفر است

(Mathematics ،Physics Fundamentals ،Philosophy of Science)

وارد صف می‌شوند. هر زمان چند کتاب به طور همزمان دارای in-degree صفر باشند، انتخاب آن‌ها بر اساس ترتیب الفبای انگلیسی انجام می‌شود.

در هر مرحله، اولین کتابِ صف خارج شده و به ترتیب نهایی مطالعه اضافه می‌شود. سپس یال‌های خروجی آن حذف می‌شوند؛ یعنی in-degree تمام کتاب‌هایی که به آن وابسته بودند، یک واحد کاهش پیدا می‌کند. اگر در نتیجه این کاهش، in-degree کتابی به صفر برسد، آن کتاب نیز به صف اضافه می‌شود.

این فرایند تا زمانی ادامه پیدا می‌کند که صف کاملا خالی شود. ترتیب خروج گره‌ها از صف، همان ترتیب نهایی Topological Sort خواهد بود.

در نهایت یک ترتیب معتبر:

Mathematics → Philosophy of Science → Physics → Quantum → Relativity → Philosophy of Mind → AI → Astrophysics → Consciousness → Theory of Everything

بخش (ب): چند ترتیب درست؟

بله ممکن است چون وقتی چند node با in-degree صفر داریم، انتخاب بین آن‌ها آزاد است. پس چند ترتیب مختلف می‌تواند درست باشد.

بخش (ج):

۵ هفته

حتی با امکان خواندن همزمان، زمان کل توسط طولانی‌ترین مسیر (Critical Path) تعیین می‌شود.

مثال طولانی‌ترین زنجیره:

Mathematics → Philosophy of Mind → Artificial Intelligence → Consciousness and Machines → Theory of Everything

۵ مرحله. هیچ‌وقت نمی‌توان این زنجیره را سریع‌تر از ۵ هفته خواند، چون هر کتاب به اتمام پیش‌نیازهایش وابسته است.

بخش ۵: سیستم پیشنهاد

در این بخش یک سیستم collaborative filtering داریم.

بخش (الف): کاربران مشابه

در این بخش، سیستم پیشنهاددهنده تلاش می‌کند برای یک کاربر جدید، بر اساس رفتار کاربران قدیمی، کتاب‌های مناسب را پیدا کند.
فرض کنید کاربر جدید کتاب «نظریه همه چیز» را خوانده و به آن امتیاز داده است. ایدهٔ اصلی سیستم این است که اگر چند کاربر دیگر نیز تقریباً همین میزان علاقه را به این کتاب نشان داده باشند، احتمال دارد سلیقهٔ آن‌ها به کاربر جدید نزدیک باشد. بنابراین سیستم از رفتار همان کاربران برای پیشنهاد کتاب استفاده می‌کند.

ابتدا باید داده‌ها بررسی شوند. در جدول امتیازها، مقدار «۵۱» برای کاربر Armin یک خطای واضح دیتایی است، چون سیستم امتیازدهی فقط بین ۰ تا ۵ تعریف شده است. بنابراین پیش از هر محاسبه، این مقدار باید به ۵ اصلاح شود. اگر این اصلاح انجام نشود، میانگین‌ها کاملا غیرواقعی می‌شوند و کل سیستم پیشنهاددهی دچار خطا خواهد شد.

سپس مقدار T از روی شماره دانشجویی محاسبه می‌شود. حالا سیستم باید کاربرانی را پیدا کند که امتیازشان به کتاب «نظریه همه چیز» به امتیاز کاربر جدید نزدیک باشد. طبق صورت سوال، تمام کاربرانی که امتیازی در بازهٔ زیر داده‌اند مشابه در نظر گرفته می‌شوند:

$$T - 0.5 \le \text{Rating} \le T + 0.5$$

اگر T = 4 باشد، بازه شباهت برابر خواهد بود با:

$$3.5 \le \text{Rating} \le 4.5$$

بنابراین فقط کاربرانی انتخاب می‌شوند که به این کتاب امتیاز ۴ داده‌اند.

پس از مشخص شدن کاربران مشابه، سیستم فقط از امتیازهای همین کاربران استفاده می‌کند و میانگین امتیاز هر کتاب دیگر را محاسبه می‌کند. دلیل این کار این است که امتیاز کاربران غیرمشابه ممکن است سلیقه‌ای کاملاً متفاوت داشته باشد و باعث خراب شدن پیشنهادها شود.

برای هر کتاب، مجموع امتیازهای کاربران مشابه محاسبه شده و سپس بر تعداد آن‌ها تقسیم می‌شود تا میانگین به دست آید. هرچه میانگین یک کتاب بیشتر باشد، یعنی کاربران مشابه آن را بیشتر دوست داشته‌اند و احتمالاً برای کاربر جدید نیز انتخاب بهتری خواهد بود.

در انتها، کتاب‌ها بر اساس این میانگین‌ها از بیشترین به کمترین مرتب می‌شوند و به همان ترتیب به کاربر جدید پیشنهاد داده می‌شوند.

بخش (ب) و بخش (ج) : پیچیدگی و trade-off

در مقیاس کوچک، این روش ساده و قابل فهم است، اما اگر سیستم میلیون‌ها کاربر و میلیون‌ها کتاب داشته باشد، اجرای همین فرآیند برای هر درخواست بسیار سنگین می‌شود.
در حالت ساده، سیستم باید تعداد زیادی کاربر را بررسی کند و سپس میانگین کتاب‌های مختلف را حساب کند. اگر تعداد کاربران را n و تعداد کتاب‌ها را m در نظر بگیریم، پیچیدگی زمانی تقریبا برابر

$O(n \times m)$ خواهد بود که در سیستم‌های واقعی بسیار پرهزینه است.

به همین دلیل سیستم‌های واقعی معمولا به جای بررسی کامل همه کاربران، از روش‌های تقریبی استفاده می‌کنند؛ روش‌هایی مثل Sampling یا Locality-Sensitive Hashing (LSH)
این روش‌ها سرعت را بسیار بیشتر می‌کنند، اما ممکن است همیشه دقیق‌ترین کاربران مشابه را پیدا نکنند. در واقع سیستم بین «دقت» و «سرعت» یک trade-off انجام می‌دهد.

در عمل، این trade-off معمولا قابل قبول است، چون هدف سیستم پیشنهاد کتاب این نیست که بهترین پیشنهاد ممکن را با دقت ریاضی مطلق پیدا کند؛ بلکه هدف این است که در زمان کوتاه، پیشنهادهایی «به اندازه کافی خوب» تولید شود تا تجربهٔ کاربر مناسب باقی بماند.

بخش ۶: نگهداری سیستم

بخش الف، ب، ج:

در Goodreads تعداد کتاب‌ها و امتیازها دائما در حال تغییر است. هر لحظه ممکن است کتاب جدیدی اضافه شود، کاربران امتیازهای جدید ثبت کنند، یا محبوبیت کتاب‌ها تغییر کند. در چنین شرایطی سیستم باید بتواند همیشه «۵ کتاب برتر» را سریع و به‌روز نمایش دهد.

اگر بخواهیم بعد از هر تغییر، کل داده‌ها را دوباره مرتب کنیم، سیستم بسیار کند و پرهزینه خواهد شد.
فرض کنید میلیون‌ها کتاب در دیتابیس وجود دارد و فقط یک امتیاز جدید ثبت شده است. اگر سیستم مجبور شود دوباره همه کتاب‌ها را sort کند، حجم بسیار زیادی از پردازش و حافظه مصرف می‌شود، در حالی که شاید فقط جایگاه یکی از کتاب‌ها تغییر کرده باشد.

مرتب‌سازی کامل داده‌ها معمولا پیچیدگی زمانی O(n log n) دارد. وقتی n بسیار بزرگ باشد، این هزینه برای هر تغییر کوچک کاملا غیرعملی می‌شود. علاوه بر آن، سیستم نمی‌تواند پاسخ لحظه‌ای بدهد و کاربران تاخیر زیادی احساس می‌کنند.

برای حل این مشکل، استفاده از Heap انتخاب بسیار مناسب‌تری است.
در این مسئله، ما فقط به ۵ کتاب برتر نیاز داریم، نه مرتب‌سازی کل میلیون‌ها کتاب.
بنابراین می‌توانیم یک Min-Heap با اندازه ثابت ۵ نگه داریم.

ایده Heap این است که همیشه فقط ۵ کتاب برتر فعلی را ذخیره کند.
کوچک‌ترین عضو Heap در ریشه قرار می‌گیرد. هر بار که کتاب جدیدی وارد می‌شود یا امتیاز کتابی تغییر می‌کند، سیستم فقط آن را با کوچک‌ترین عضو Heap مقایسه می‌کند.
اگر امتیازش کمتر باشد، اصلا وارد Heap نمی‌شود. اگر امتیازش بیشتر باشد، کوچک‌ترین عضو حذف شده و کتاب جدید وارد Heap می‌شود. به این ترتیب، سیستم بدون مرتب‌سازی کل داده‌ها همیشه می‌تواند ۵ کتاب برتر را نگه دارد.

درج یا حذف در Heap پیچیدگی O(log k) دارد که در اینجا k=5 است و مقدار بسیار کوچکی محسوب می‌شود.
به همین دلیل این روش نسبت به مرتب‌سازی کامل بسیار سریع‌تر و بهینه‌تر است.

بخش د، و، ه :

کاربر جدیدی که هنوز هیچ فعالیتی در سیستم ندارد؛ یعنی:

هیچ کتابی نخوانده است، هیچ امتیازی ثبت نکرده است، و سیستم هیچ اطلاعاتی از سلیقه او ندارد.

این مشکل در سیستم‌های پیشنهاددهنده با نام Cold Start Problem شناخته می‌شود.

روش‌هایی مثل Collaborative Filtering بر اساس شباهت کاربران کار می‌کنند.
یعنی سیستم باید رفتار کاربر جدید را با رفتار کاربران دیگر مقایسه کند تا افراد مشابه را

پیدا کند. اما وقتی کاربر جدید هیچ داده‌ای ندارد، عملا چیزی برای مقایسه وجود نخواهد داشت. به همین دلیل سیستم در ابتدای کار نمی‌تواند از روش‌های مبتنی بر شباهت استفاده کند.

در چنین شرایطی معمولا از روش‌های ساده‌تر و عمومی‌تر استفاده می‌شود؛ مثلا:

- نمایش محبوب‌ترین کتاب‌های ماه،
- کتاب‌های ترند،
- پرفروش‌ترین آثار،
- یا کتاب‌هایی که بیشترین امتیاز را گرفته‌اند.

این روش‌ها اگرچه شخصی‌سازی‌شده نیستند، اما برای شروع تجربه کاربر مناسب‌اند و کمک می‌کنند کاربر اولین تعامل‌های خود را با سیستم انجام دهد. بعد از اینکه چند کتاب خواند یا چند امتیاز ثبت کرد، سیستم کم‌کم می‌تواند سلیقه او را یاد بگیرد و پیشنهادهای شخصی‌تری ارائه دهد.

بخش ۷: تقلب و تعصب در سیستم امتیازدهی

سیستم امتیازدهی Goodreads قرار است نشان دهد کاربران واقعا چه نظری دربارهٔ کتاب‌ها دارند. اما در عمل، رفتار کاربران همیشه کاملا منصفانه و منطقی نیست. بعضی کاربران ممکن است به دلیل علاقه شدید به یک نویسنده، بدون قضاوت واقعی به تمام کتاب‌های او امتیاز کامل بدهند. برخی دیگر ممکن است از نویسنده‌ای خوششان نیاید و عمدا به همهٔ آثار او امتیاز بسیار پایین بدهند. گاهی حتی کاربران بدون اینکه کتابی را کامل خوانده باشند، صرفا از روی تعصب یا هیجان به آن امتیاز می‌دهند. این رفتارها باعث می‌شود داده‌های سیستم آلوده شوند و میانگین امتیازها دیگر تصویر دقیقی از کیفیت واقعی کتاب‌ها نباشد.

اگر فقط از میانگین ساده استفاده کنیم، سیستم به‌راحتی فریب می‌خورد.
مثلا فرض کنید:

کتاب A فقط ۲ رأی داشته و هر دو نفر امتیاز ۵ داده‌اند، پس میانگین آن ۵ می‌شود.

کتاب B هزاران رأی داشته و میانگین آن ۴.۸ است.

اگر صرفا میانگین ساده را مقایسه کنیم، کتاب A بهتر از کتاب B به نظر می‌رسد، در حالی که این نتیجه منطقی نیست؛ چون فقط دو نفر درباره کتاب A نظر داده‌اند و ممکن است هر دو رأی غیرواقعی یا biased باشند.

برای حل این مشکل، سیستم‌های واقعی معمولا از Bayesian Average یا میانگین بیزی استفاده می‌کنند. ایده اصلی میانگین بیزی این است که وقتی تعداد رأی‌های یک کتاب کم باشد، سیستم نباید بیش از حد به میانگین آن اعتماد کند. بنابراین میانگین کتاب را کمی به سمت میانگین کل سایت نزدیک می‌کند تا اثر رأی‌های محدود کمتر شود.
در نتیجه

کتابی با ۱ رأی ۵ ستاره فورا به رتبه اول نمی‌رسد،

اما کتابی که هزاران رأی با میانگین بالا دارد، جایگاه پایدارتری خواهد داشت.

این روش تا حد زیادی جلوی تقلب و نوسانات شدید را می‌گیرد، اما خودش یک مشکل جدید ایجاد می‌کند.

کتاب‌های تازه‌منتشرشده معمولا رأی کمی دارند. حتی اگر واقعا کتاب فوق‌العاده‌ای باشند، میانگین بیزی آن‌ها را به میانگین عمومی سایت نزدیک می‌کند و رتبه پایینی به آن‌ها می‌دهد. در نتیجه کتاب‌های جدید فرصت دیده شدن پیدا نمی‌کنند و کتاب‌های قدیمی و مشهور دائما در صدر باقی می‌مانند. اینجاست که trade-off اصلی سیستم مشخص می‌شود:

اگر بیش از حد به رأی‌های کم اعتماد کنیم، سیستم در برابر تقلب آسیب‌پذیر می‌شود.
اگر بیش از حد محتاط باشیم، کتاب‌های جدید عملا شانس رشد پیدا نمی‌کنند.

به همین دلیل سیستم‌های واقعی معمولا فقط به یک روش تکیه نمی‌کنند و از ترکیب چند ایده استفاده می‌کنند.

یکی از روش‌های رایج استفاده از Time Decay است.
در این روش، رأی‌های جدیدتر وزن بیشتری نسبت به رأی‌های خیلی قدیمی دارند. این کار کمک می‌کند کتاب‌های تازه بتوانند سریع‌تر رشد کنند و سیستم فقط اسیر محبوبیت‌های قدیمی نشود.

همچنین بسیاری از سیستم‌ها از Hybrid Approach استفاده می‌کنند؛ یعنی چندین معیار مختلف را هم‌زمان ترکیب می‌کنند، مثل:

- میانگین بیزی،
- تعداد رأی‌ها،
- تازگی کتاب،
- رفتار کاربران معتبر،
- و حتی الگوهای غیرعادی رأی‌دادن.

هدف نهایی این است که سیستم هم در برابر تقلب مقاوم باشد، هم کیفیت واقعی کتاب‌ها را بهتر تشخیص دهد، و هم به آثار جدید فرصت دیده شدن بدهد.