**خلاصه SequenceMatcher چطور کار می‌کند؟**

SequenceMatcher رشته‌ها یا لیست‌ها را با هدف یافتن **بیشترین بخش‌های مشابه پشت‌سر‌هم (Longest Matching Blocks)** مقایسه می‌کند و شباهت آن‌ها را بر اساس **تعداد و طول این بلوک‌های مشابه** تخمین می‌زند.

**پایه نظری: الگوریتم LCS**

* الگوریتم کلاسیک LCS (بزرگ‌ترین زیر دنباله مشترک) به دنبال یافتن دنباله‌ای از عناصر مشابه در هر دو رشته است که ترتیب آن‌ها حفظ شده باشد، اما لازم نیست پشت‌سر‌هم باشند.
* زمان اجرای LCS کلاسیک: O(n \* m) برای دو رشته با طول‌های n و m.

**SequenceMatcher چطور متفاوت است؟**

**1. شناسایی "matching blocks"**

SequenceMatcher به دنبال **بزرگ‌ترین بلوک‌های متوالی یکسان** در دو رشته است. این بلوک‌ها پشت سر هم هستند و LCS نیستند، بلکه "matching block" هستند.

**2. پیاده‌سازی هوشمند با الگوریتم “Greedy Recursive Matching”**

پیاده‌سازی آن به‌شکل بازگشتی و greedy است:

* بزرگ‌ترین بلاک مشابه بین a و b پیدا می‌شود.
* سپس فضای قبل و بعد از این بلاک جدا شده و به‌صورت بازگشتی بررسی می‌شود.
* این فرایند تا جایی ادامه می‌یابد که هیچ بلاک مشابهی با اندازه بزرگ‌تر از یک پیدا نشود.

**3. استفاده از دیکشنری برای بهینه‌سازی )**سرعت بیشتر(

* رشته کوتاه‌تر (مثلاً b) به‌صورت یک دیکشنری از کاراکتر → موقعیت‌ها ذخیره می‌شود.
* سپس هنگام جستجو، بلاک‌های مشابه در a بررسی می‌شوند و موقعیت‌های تطابق در b سریعاً از دیکشنری پیدا می‌شوند.
* این کار باعث صرفه‌جویی قابل‌توجه در زمان نسبت به الگوریتم LCS خام می‌شود.

**4. پشتیبانی از توابع isjunk**

SequenceMatcher می‌تواند از یک تابع فیلتر برای حذف نویز (مثلاً فاصله، نقطه، یا تکرار حروف بی‌معنا) استفاده کند:

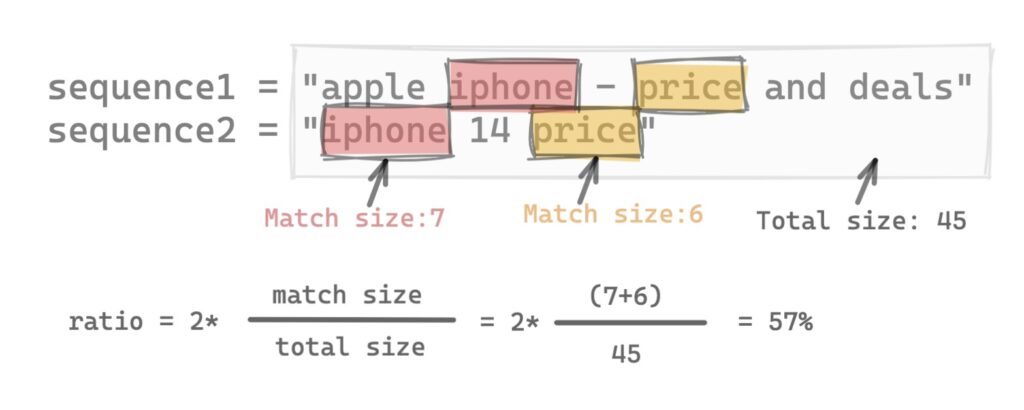
SequenceMatcher(isjunk=my\_junk\_filter, a, b)

**📐 محاسبه نسبت شباهت .ratio()**

پس از یافتن تمام matching block‌ها، نسبت شباهت طبق فرمول زیر محاسبه می‌شود:

ratio=2×Mlen(a)+len(b)\text{ratio} = \frac{2 \times M}{\text{len}(a) + \text{len}(b)}

* M: تعداد کاراکترهای کل موجود در matching blockها (یعنی کل کاراکترهای مشترک پشت سر هم)
* len(a), len(b): طول رشته‌های اصلی



**📦 سایر توابع کاربردی SequenceMatcher**

| **تابع** | **کاربرد** |
| --- | --- |
| .ratio() | محاسبه نسبت شباهت بین a و b |
| .get\_matching\_blocks() | لیست تمام بلوک‌های مشابه |
| .get\_opcodes() | لیست عملیات لازم برای تبدیل a به b مانند (diff) |
| .quick\_ratio() | نسخه‌ی سریع‌تر ولی تقریبی .ratio() |
| .real\_quick\_ratio() | حتی سریع‌تر (برای فیلتر اولیه) |

**⏱ پیچیدگی زمانی**

* در بدترین حالت تقریباً O(n \* log n) به دلیل استفاده هوشمند از دیکشنری و بازگشت‌های سریع.
* در عمل بسیار سریع‌تر از LCS معمولی است.

**🎯 کاربردها در دنیای واقعی**

* تشخیص تقلب یا شباهت در پاسخ‌ها
* الگوریتم‌های تصحیح خودکار (مثل ویرایش املایی)
* مقایسه فایل‌های متنی (diff tools)
* تطبیق تقریبی در جستجوگرها
* ابزارهای ترجمه یا مقایسه کد