تعداد متمایز در مجموعه داده با مقادیر تکراری

محسن هوشمند

فهرست

تعداد منحصربه‌فرد

مسئله تخمین تعداد منحصربه‌فرد در پی یافتن تعداد مقادیر متمایز در مجموعه داده‌ای دارای اعضای تکراری است که در حوزه‌هائی گوناگون تولید نرم‌افزار اعم از پایگاه داده‌ها و ترافیک شبکه کاربرد دارد. در روش سنتی و دم دستی تعیین تعداد دقیق یک مجموعه، فهرستی از همه مقادیر ایجاد، و سپس با مرتب‌سازی و جستجو از شمردن عضو تکراری اجتناب و در نتیجه تعداد مقادیر متمایز احصاء می‌شود. ، هنگامی که تعداد کل اعضای مجموعه از جمله تکراری‌ها برابر Nباشد، با روش مذکور پیچیدگی زمانی شمارش دقیق مقادیر منحصربه‌فرد برابر با $O(N \log N)$ است و نیاز به حافظه کمکی الزامی است. اما با گسترش خدمات اینترنتی، با میلیاردها کلیک، جستجو، و سفارش که روزانه با تعداد بسیار کمتری کاربر انجام می‌پذیرد، بازطراحی شمارش تعداد متمایز را اهمیتی دوباره بخشیده است. خاصه، تعیین تعداد منحصر به فرد با پیمایش یکبارهٔ داده و در فضای اشغالی کمتر از حوزه‌های مهم تحقیقاتی به شمار می‌رود.

در حال حاضر، تخمین کاردینالیتی برای تعیین تعداد بازدیدکنندهٔ یکتا از محصولی خاص، تعداد کاربران متمایز استفاده کننده از قابلیتی مشخص در وب‌اپلیکیشن، و همچنین برای تشخیص تغییرات ناگهانی در تعداد آدرس‌های IP مبدأ–مقصدِ یکتا که از یک روتر عبور می‌کنند (که ممکن است نشان‌دهندهٔ حملهٔ انکارِ خدمت باشد) به‌کار می‌رود.

به‌علاوه، به‌دلیل نحوهٔ تکثیر مکرر اطلاعات در وب، اندازه‌گیری کاردینالیتی در تشخیص تعداد محتوای متمایز برای مثال، تعداد مقاله‌های خبری یکتا یا تعداد نسخه‌های متفاوت از محتوای وب‌سایت یاری‌رسان است. نظر به مجموعه داده‌های عظیم امروزی، علاقهٔ فزاینده‌ای به طراحی الگوریتم‌هایی وجود دارد که بتوانند تعداد منحصر به فرد مجموعه را با دقتی مناسب و در فضایی بسیار کوچک‌تر از اندازهٔ خود مجموعه تقریب بزنند.

مثال ۱- بازدیدکنندگان منحصربه‌فرد

یکی از شاخص‌های کلیدی ارزشمند KPIs برای هر تارمانه، تعداد بازدیدکنندگان منحصربه‌فرد آن در بازهٔ زمانی مشخص است. جهت سادگی، فرض می‌کنیم که هر بازدیدکننده از یک نشانی IP در تمامی بازدیدهایش استفاده می‌کند، در نتیجه، با شمارش تعداد IP-های منحصربه‌فردکه با پروتکل اینترنت نسخه IPv٦ با رشته‌های ۱۲۸ بیتی نشان داده می‌شوند تعداد بازدیدکنندگان احصاء می‌شود. آیا این کار آسانی است؟ آیا می‌توان با تکیهٔ صرف بر روش‌های کلاسیک برای شمارش دقیق تعداد را بدست آورد؟ پاسخ به این سوال به میزان استفاده از تارمانه بستگی دارد.

مطابق SimilarWeb در گزارش آمار ترافیک مارس ۲۰۱۷ (اسفند ۱۳۹۵) سه تارمانهٔ خرده فروشی پر مشتری در ممالک اتازونی اعم از آمازون و ائی‌بی و والمارت، میانگین تعداد بازدید از این تارمانه‌ها حدود ۱.۴۴ میلیارد و میانگین تعداد صفحات مشاهده شده در هر بازدید ۸.۲۴ بود. سخن کوتاه، آمار اسفند ۱۳۹۵ مراجعه به حدود ۱۲ میلیارد آدرس IP ۱۲۸ بیتی است و نگه‌داری همه آنها اندازه‌ای در حدود ۱۹۲ گیگابایت لازم دارد. اگر فرض کنیم که از هر ۱۰ بازدیدکننده، یک نفر منحصربه‌فرد بوده است، می‌توانیم انتظار داشته باشیم که تعداد منحصربه‌فرد چنین مجموعه‌ای حدود ۱۴۴ میلیون باشد و حافظه مورد نیاز برای ذخیره فهرست مقادیر منحصربه‌فرد ۲۳ گیگابایت است.

مثال زیر چالش تخمین تعداد منحصربه‌فرد را برای تحقیقات علمی نشان می‌دهد.

مثال ۲- تحلیل DNA

مطالعه همبستگی‌ها در توالی‌های DNA، از پژوهش‌های مهم در تحقیقات ژنوم انسانی است. مولکول‌های DNA شامل دو رشتهٔ جفت شده هستند که هر کدام از چهار واحد شیمیایی پایه DNA، با علامت A (آدنین)، G (گوانین)، C (سیتوزین) و T (تیمین) تشکیل شده‌اند. ژنوم انسان حدود سه میلیارد جفت پایه دارد. توالی‌یابی به معنای تعیین ترتیب دقیق جفت‌های پایه در قطعاتی از

DNA است. از دید ریاضی، توالی DNA را می‌توان به عنوان رشته‌ای با اندازه‌های طولانی از نمادهای A، G، C، T در نظر گرفت، به طوری که می‌توان آنها را به عنوان نمونه‌ای از مجموعه داده‌ای بالقوه بی‌نهایت در نظر گرفت. مسئلهٔ اندازه‌گیری همبستگی در پی تعیین تعداد زیررشته‌های متمایز با اندازه ثابت در قطعه‌ای DNA تعریف می‌شود. مفروض مسئله مذکور این است که هر توالی با زیررشته‌های متمایز کم، همبستگی بیشتری نسبت به توالی دیگری با اندازه مشابه اما با زیررشته‌های متمایز بیشتر دارد. چنین آزمایش‌هایی نیاز به اجرای چندباره روی فایل‌های بزرگ متعدد دارند و برای تسریع تحقیقات به حافظه محدود یا حتی ثابت و زمان اجرای کم دسترسی و تاکید دارند، که امر مذکور با الگوریتم‌های شمارش دقیق ممکن نیست.

بنابراین، تخمین دقیق تعداد منحصربه‌فرد به زمان پردازش و حافظه زیاد نیاز دارند. برنامه‌های داده بزرگ به سمت رویکردهای عملی‌تر غالبا بر مبنای الگوریتم‌های احتمالاتی و عرضهٔ پاسخی تقریبی گرایش دارند.

در خور ذکر است که هنگام پردازش داده‌ها، داشتن تحلیلی از اندازه مجموعه داده ورودی و تعداد مقادیر متمایز احتمالی در تصمیم‌گیری نقشی پررنگ دارد. دنبالهٔ بی‌نهایت (بالقوه بی‌نهایت) رشته‌های تک حرفی a، d، s، ... را در نظر می‌گیریم که بر اساس حروف الفبای لاتین باشد. تعداد منحصربه‌فرد را می‌توان به راحتی تخمین زد و کران بالای آن به تعداد حروف لاتین محدود می‌شود. بدیهی است که در این مورد، نیازی به استفاده از هیچ رویکرد احتمالاتی نیست و راه‌حل مبتنی بر لغت‌نامهٔ ساده برای محاسبه دقیق تعداد منحصربه‌فرد بسیار خوب عمل می‌کند.

در حل مسئله تعداد منحصربه‌فرد، بسیاری از روش‌های احتمالاتی مطرح از الگوریتم فیلتر بلوم تاثیر گرفته و بر اساس مقادیر درهم عمل کرده، سپس، الگوهای رایج در توزیع آنها را مشاهده می‌کنند و «حدس‌های» منطقی در مورد تعداد مقادیر منحصربه‌فرد بدون نیاز به ذخیره همه آنها می‌زنند.

شمارش اقلام متمایز در پایگاه‌های داده

احتمالا آشناترین مثال برای سنجش کاردینالیته (اندازهٔ مجموعه) از پایگاه‌های داده و نحوهٔ استفادهٔ SQL از کلیدواژهٔ DISTINCT ناشی می‌شود. وقتی این کلیدواژه بر روی ستونی در جدول اعمال شود، دستور SELECT DISTINCT تمام اقلام متمایز آن ستون را و دستور SELECT COUNT DISTINCT تعداد اقلام متمایز موجود در ستون مورد نظر را برمی‌گرداند. پرس‌وجوهای حاوی COUNT DISTINCT در بسیاری از حوزه‌ها به‌ویژه در حوزهٔ تجارت الکترونیک یا زمانی که آمار استفاده از تارمانه موردنیاز باشد بسیار شایع و رایج است. داده‌های بازدید کاربران اغلب در جدولی به نام «بازدیدهای-روزانه» ثبت می‌شوند که معمولا بسیار بزرگ است و دارای ویژگی‌هایی نظیر session_id، timestamp، product_id، user_ip_address، visit_duration و غیره است. با اجرای دستور انتخاب زیر:

```
SELECT COUNT (DISTINCT user_ip_address) WHERE product_id = ۹۸۷۳۹۴۷
FROM DAILY_VISITS
```

تعداد آدرس‌های IP متمایز (یعنی کاربران) دسترسی‌یافته به محصول با شناسهٔ ۹۸۷۳۹۴۷ را در روزی معین دریافت خواهیم کرد. در تارمانهٔ شلوغ، جدول بازدیدهای روزانه می‌تواند تا چند میلیارد ردیف رشد کند، و این پرس‌وجوی خاص ممکن است نیاز به زمان زیادی برای پاسخ داشته و با تأخیر همراه باشد.

این تأخیر عمدتا ناشی از عملیات مرتب‌سازی است که COUNT DISTINCT کلاسیک در بیشتر پایگاه‌های داده (مانند Azure SQL/SQL Server) اعمال می‌کنند، مگر آنکه ستون از قبل مرتب شده باشد. پس از مرتب‌سازی ستون، تمام رکوردهای تکراری در کنار یکدیگر قرار می‌گیرند، و یک پیمایش ترتیبی کافی است تا اقلام متمایز شناسایی و شمارش شوند. هزینهٔ عملیات مرتب‌سازی در جدولی با n ردیف، $O(n\ logn)$ است و نه تنها برای چند میلیارد ردیف بلکه حتی برای چند میلیون ردیف به خوبی مقیاس‌پذیر نیست، حتی پرس‌وجوهای ساده نیز COUNT DISTINCT-ها و GROUP BY-های متعددی روی ستون‌های مختلف انجام می‌دهند، و مرتب‌سازی یک ستون به کاهش پیچیدگی مرتب‌سازی ستون دیگر کمکی نمی‌کند. جهت تسریع کار می‌توان از جدول درهم استفاده کرد، اما جدول درهم همچنان به فضایی خطی نسبت به تعداد مقادیر متمایز k نیاز دارد. از آنجا که k می‌تواند تا n افزایش یابد، استفاده از درهم‌سازی نیز مقرون به صرفه نیست.

جهت رفع مسائل مقیاس‌پذیری، نسخه‌های جدیدتر سیستم‌های مدیریت پایگاه‌داده و انباره‌های داده به تخمین تعداد منحصر به فرد روی آورده‌اند. محیط $SQL\ Server$ ۲۰۱۹ دارای عملیات APPROX_COUNT_DISTINCT (http://mng.bz/QWjm) است که از فضای بسیار کمی استفاده و سریع کار می‌کند. Google BigQuery این رویکرد تقریبی و احتمالاتی را در COUNT DISTINCT به عنوان پیش‌فرض قرار می‌دهد، و EXACT_COUNT_DISTINCT را برای موقعیت‌هایی که مطلقا به پاسخ دقیق نیاز است، محفوظ نگه می‌دارد (PJ۴http://mng.bz/y). الگوریتمی که در چنین تخمین‌گرهائی اجرا می‌شود ابرلگ‌لگ نام دارد و آن را فلاژوله و همکاران [۲] ابداع کردند، به طوری که صرفه‌جویی سخت زیادی در فضا (در حد کیلوبایت) ارائه می‌دهد، در حالی که مجموعه‌داده‌هایی با تعداد اعضای در حد تریلیون را پردازش می‌کند و میزان خطا را نسبتا پایین در مرتبه‌ای $O(\frac{۱}{\sqrt{m}})$ نگه می‌دارد که m نشان‌دهندهٔ تعداد مدخل‌های حافظهٔ چند بیتی است. انتخاب رایج برای m مقدار $۲^{۱۴}$ است.

نمونه‌های متعددی از صرفه‌جویی در فضا به بهای کاهش دقت معرفی شده‌اند. با این حال، ابرلگ‌لگ معنای تازه‌ای به کارایی فضا می‌بخشد، به‌گونه‌ای که تقریبا همیشه در محدودهٔ چند کیلوبایت باقی می‌ماند و در عین حال تخمینی از تعداد منحصر به فرد واقعی را با میانگین خطای کوچک (مثلا ± ۲ درصد) به دست می‌دهد. در ادامه، چند روش تخمین تعداد منحصر به فرد را معرفی می‌کنیم.

شمارش خطی

الگوریتم شمارش خطی به عنوان اولین رویکرد احتمالاتی برای مسئله تعداد منحصربه‌فرد مطرح شد. طرح اصلی را مورتون آستراهان، ماریو شولنیک، و کیو-یانگ وانگ در سال ۱۳۶۶ معرفی کردند و الگوریتم عملی را کیو-یانگ وانگ، براد واندر-زاندن، و هوارد تیلور در سال ۱۳۶۹پیشنهاد دادند. طرح اصلی پیشنهادی آنها استفاده از درهم مقادیر با تابعی درهم‌ساز است که موجب برتری نسبت به روش‌های دقیق شده است. استفاده از درهم امکان حذف تکراری‌ها بدون نیاز به مرتب‌سازی مقادیر را فراهم می‌کند. البته این فرصت بدون هزینه نیست و از دقت پاسخ می‌کاهد و احتمال خطا به دلیل تصادم‌های احتمالی درهم (نمی‌توان تکراری‌ها خاصه «تکراری‌های تصادفی» را تشخیص داد) را موجب می‌شود. سخن کوتاه، استفاده از چنین جدول درهمی به همراه روش پیمایش مناسب منجر به بهبود عملکرد فضا و زمان نسبت به روش کامل یا جستجو با کمک درهم می‌شود.

با این حال، برای مجموعه‌های داده با تعداد منحصربه‌فرد عظیم، چنین جدول‌های درهمی می‌توانند بسیار بزرگ باشند و به حافظه‌ای نیاز دارند که به صورت خطی با تعداد مقادیر متمایز در مجموعه رشد می‌کند. برای سیستم‌هایی با حافظه محدود، در نقطه‌ای به ذخیره‌سازی دیسک یا توزیع شده نیاز خواهد داشت، که به دلیل دسترسی کند به دیسک یا شبکه، مزایای جدول‌های درهم سخت کاهش می‌یابد.

مشابه ایده فیلتر بلوم، برای دور زدن چنین مشکلی، الگوریتم شمارش خطی مقادیر درهم را ذخیره نمی‌کند، بلکه بیت‌های متناظر آنها را ذخیره می‌کند و آرایهٔ بیتی (تحت نام «شمارندهٔ خطی») را جانشین جدول درهم می‌کند. فرض می‌شود که مقدار m با تعداد مقادیر متمایز مورد انتظار n متناسب است، اما برای اکثر موارد فقط به یک بیت به ازای هر مقدار نیاز دارد که قابل اجرا است. در ابتدا، همه بیت‌ها در «شمارندهٔ خطی» برابر با صفر هستند. با افزودن مقدار جدید x به چنین داده‌ساختاری، مقدار درهم‌ساز h(x) متناظرش محاسبه و مقدار بیت در نمایهٔ متناظر در شمارنده برابر یک می‌شود.

الگوریتم ۱- درج در شمارنده خطی
ورودی: مقدار $x \in D$
ورودی: شمارنده خطی با تابع درهم‌ساز h
$j \leftarrow h(x)$
اگر LINEARCOUNTER[j] = ۰ آنگاه
$LINEARCOUNTER[j] \leftarrow ۱$

به دلیل استفاده از تک تابع درهم‌ساز h، امکان دارد دو مقدار ورودی متفاوت منجر به مقداردهی یک بیت در آرایه می‌شود و در نتیجه به تعداد تصادم‌های فراوانی منجر شود. بنابراین، تعداد دقیق (یا حتی نزدیک به دقیق) مقادیر متمایز دیگر نمی‌تواند مستقیما از چنین طرح فشرده‌ای به دست آید.

الگوریتم منجر به توزیع مقادیر در سطل‌ها (بیت‌های نمایه شدهٔ مقادیر درهم) می‌شود و آرایه بیتی نمایشگر موجب می‌شود مدخل‌های متناظر مقداردهی شوند. مشاهده تعداد یک‌ها در آرایه تخمین تعداد منحصربه‌فرد را به دست می‌دهد.

در مرحله اول الگوریتم شمارش خطی، داده‌ساختار را همانطور که در الگوریتم ۱ نشان داده شد، ایجاد کمی‌کند. با داشتن چنین طرح فشرده‌ای، تعداد منحصربه‌فرد را می‌توان با استفاده از کسر مشاهده شده بیت‌های خالی V با معادله تخمین زد:

$$n \approx -m \cdot \ln V \qquad (۱)$$

تصادم‌ها بر تخمین تعداد منحصربه‌فرد در الگوریتم شمارش خطی تأثیر می‌گذارند. به بیان دیگر، هر تصادم تعداد بیت‌هایی که باید تنظیم شوند را کاهش می‌دهد و کسر مشاهده شده بیت‌های تنظیم نشده را بزرگتر از مقدار واقعی می‌کند. اگر تصادم درهم وجود نداشت، تعداد نهایی بیت‌های تنظیم شده، تعداد منحصربه‌فرد مورد نظر بود. با این حال، تصادم‌ها اجتناب‌ناپذیر هستند و معادله (۱) در واقع تخمین بیش از حدی از تعداد دقیق منحصربه‌فرد ارائه می‌دهد و از آنجایی که تعداد منحصربه‌فرد مقداری عدد صحیح است، ترجیح می‌دهیم نتیجه آن را به نزدیک‌ترین عدد صحیح کوچکتر گرد کنیم. شبیه محاسبات یافتن تعداد منحصر به فرد در الگوریتم بلوم می‌توان مقدار تخمین معادله (۱) را به صورت زیر به دست آورد. احتمال صفر بودن بیتی پس از درج n عضو جدید برابر است با

$$\text{pr (bit} = ۰) = \left(۱ - \frac{۱}{m}\right)^n \approx e^{-n/m}$$

در نتیجه، میانگین نسبت یک‌ها برابر است با

$$\frac{E[N]}{m} = ۱ - e^{-n/m}$$

و N تعدد بیت‌های متناظر یک در فیلتر است. جهت تقریب n از N می‌توان از $N/m = ۱ - e^{-n/m}$ مقدار زیر را بدست آورد:

$$e^{-n/m} = ۱ - \frac{N}{m}$$

$$e^{-n/m} = ۱ - \frac{N}{m}$$

لگاریتم طبیعی دو طرف را محاسبه می‌کنیم

$$-\frac{n}{m} = \ln\left(1 - \frac{N}{m}\right)$$

با قرار دادن $V = 1 - \frac{N}{M}$ داریم

$$n = -m \ln V$$

که دقیقا برابر مقداری است که الگوریتم وقتی $0 < N < m$ برمی‌گرداند. سخن کوتاه، الگوریتم کامل شمارش را به صورت زیر تدیون می‌کنیم.

الگوریتم ۲- تخمین تعداد منحصربه‌فرد با شمارش خطی

ورودی: مجموعه داده D

خروجی: تخمین تعداد منحصربه‌فرد

$LINEARCOUNTER[i] \leftarrow 0, i = 0 \dots m - 1$

برای هر $x \in D$ انجام بده

$Add(x, LINEARCOUNTER)$

$Z \leftarrow count_{i=0 \dots m-1} (LINEARCOUNTER[i] = 0)$

برگرداندن $\left\lfloor -m \times \ln \frac{Z}{m} \right\rfloor$

مثال ۳- الگوریتم شمارش خطی: مجموعه داده‌ای شامل بیست نام از پایتخت‌ها **برلین**، برلین، **پاریس**، برلین، **لیسبون**، **کیف**، پاریس، **لندن**، **رم**، **آتن**، **مادرید**، **وین**، رم، رم، لیسبون، برلین، پاریس، لندن، کیف، **واشنگتن** را در نظر می‌گیریم. برای چنین تعداد منحصربه‌فرد کوچک (تعداد دقیق منحصربه‌فرد ۱۰ است) برای داشتن خطای استاندارد حدود ده درصد نیاز است طول داده‌ساختار را حداقل به اندازه تعداد مقادیر منحصربه‌فرد مورد انتظار انتخاب کنیم، بنابراین $m = 2^4$ را انتخاب می‌کنیم. به عنوان تابع درهم‌ساز h با مقادیر در $\{0, 1, \dots, 2^4 - 1\}$ از تابعی مبتنی بر مورمور۳ نوع ۳۲ بیتی استفاده می‌کنیم که به صورت زیر تعریف شده است:

$$h(x) = MurmurHash3(x) \% m$$

و مقادیر درهم شهرها را می‌توان در جدول زیر یافت.

شهر	h(شهر)	شهر	h(شهر)
آتن	۱۲	لندن	۱۴
برلین	۷	لیسبون	۱۵
پاریس	۸	مادرید	۱۴
رم	۱	واشنگتن	۱۱
کیف	۱۳	وین	۶

همان‌طور که در جدول مشخص است، شهرهای لندن و مادرید مقدار درهم یکسانی دارند، که البته همان‌طور که ذکر شد چنین تصادم‌هایی مورد انتظار و کاملا طبیعی هستند. داده‌ساختار نمای زیر را دارد:

۰	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱	۱۲	۱۳	۱۴	۱۵
۰	۱	۰	۰	۰	۰	۱	۱	۱	۰	۰	۱	۱	۱	۱	۱

با توجه به الگوریتم شمارش خطی، کسر V بیت‌های خالی را در LINEARCOUNTER محاسبه می‌کنیم:

$$V = \frac{۹}{۱۶} = ۰.۵۶۲۵$$

و تعداد منحصربه‌فرد تخمین زده شده برابر $n \approx -۱۶ \times \ln ۰.۵۶۲۵ \approx ۹.۲۰۶$ است.

در ادامه چند ویژگی الگوریتم شمارنده خطی را برمی‌شماریم.

در صورتی که تابع درهم‌ساز h دارای زمان محاسبه‌ای ثابت باشد (که برای غالب توابع درهم‌ساز صادق است)، زمان پردازش هر مقدار برابر مرتبهٔ O(N) است، که در آن N تعداد کل مقادیر از جمله تکراری‌ها است. بنابراین، الگوریتم دارای پیچیدگی زمانی O(N) است.

مانند هر الگوریتم احتمالاتی دیگر، تعدادی پارامتر وجود دارد که می‌توان برای تأثیرگذاری بر عملکرد آن تنظیم کرد.

دقت مورد انتظار تخمین به اندازهٔ آرایهٔ بیتی m و نسبت آن به تعداد مقادیر متمایز $\alpha = \frac{m}{n}$ که ضریب بار نامیده می‌شود، بستگی دارد. مگر اینکه $\alpha \geq ۱$ ($m > n$ که در عمل جالب نیست)، احتمالی غیرصفر $pr_{\text{اشباع}}$ وجود دارد که آرایهٔ بیتی پر شود، که احتمال پر شدن یا اشباع نامیده می‌شود و نتایج الگوریتم را کاملا مخدوش کرده و مقدار معادلهٔ (۱) را بی‌نهایت می‌کند. احتمال $pr_{\text{اشباع}}$ به ضریب بار و در نتیجه به اندازهٔ m بستگی دارد. پس باید به اندازهٔ کافی بزرگ انتخاب شود تا احتمال پر شدن ناچیز باشد.

خطای استاندارد δ معیاری از تغییرپذیری تخمین حاصل از شمارش خطی است که تعادلی بین آن و اندازه آرایه بیتی m وجود دارد. کاهش خطای استاندارد منجر به تخمین‌های دقیق‌تر می‌شود، اما حافظه موردنیاز را افزایش می‌دهد.

جدول ۲. تعادل بین دقت و اندازه آرایه بیتی

n	m	
	$\delta = ۱٪$	$\delta = ۱۰٪$
۱۰۰۰	۵۳۲۹	۲۶۸
۱۰۰۰۰	۷۹۶۰	۱۷۰۹
۱۰۰۰۰۰	۲۶۷۲۹	۱۲۷۴۴
۱۰۰۰۰۰۰	۱۵۴۱۷۱	۱۰۰۸۸۰
۱۰۰۰۰۰۰۰	۱۰۹۶۵۸۲	۸۳۱۸۰۹
۱۰۰۰۰۰۰۰۰	۸۵۷۱۰۱۳	۷۰۶۱۷۶۰

وابستگی به انتخاب m بسیار پیچیده است و راه‌حل تحلیلی ندارد. با این حال، نویسندگان الگوریتم مقادیر از پیش محاسبه شده‌ای برای احتمال قابل قبول تکمیل $pr_{\text{اشباع}} = ۰.۷٪$ را معرفی کرده‌اند و می‌توان از آنها به عنوان مرجع استفاده کرد. به دلیل صفر نبودن احتمال اشباع، آرایه بیتی به ندرت پر می‌شود و الگوریتم ۲ را مخدوش می‌کند. هنگام کار با مجموعه‌های داده کوچک، می‌توان همه مقادیر را با تابع درهم‌ساز متفاوت دوباره نمایه کرد یا اندازهٔ شماندهٔ خطی را افزایش داد. اما، چنین راه‌حل‌هائی برای مجموعه‌های داده عظیم کار نخواهند کرد و همراه با پیچیدگی زمانی نسبتا بالا، نیاز به جستجوی جانشین‌ها دارند.

با این حال، شمارش خطی زمانی که تعداد منحصربه‌فرد مجموعه داده‌ای که اندازه‌گیری می‌شود خیلی بزرگ نیست، بسیار خوب عمل می‌کند و می‌توان از آن برای بهبود الگوریتم‌های دیگر استفاده کرد که برای ارائه بهترین رفتار ممکن برای تعداد منحصربه‌فردهای عظیم ایجاد شده‌اند. در الگوریتم شمارش خطی، تخمین تعداد منحصربه‌فرد تقریبا متناسب با مقدار دقیق است، به همین دلیل از

اصطلاح «خطی» استفاده می‌شود. در بخش بعدی، الگوریتم جانشینی را در نظر می‌گیریم که می‌تواند به عنوان شمارش «لگاریتمی» طبقه‌بندی شود زیرا مبتنی بر تخمین‌هایی است که لگاریتم تعداد منحصربه‌فرد واقعی هستند.

شمارش احتمالاتی

فیلیپ فلاژوله و جی. نایجل مارتین در سال ۱۳۶۴ الگوریتم شمارشی بر اساس مشاهده الگوهای متداول در نمایش‌های درهم مقادیر ورودی معرفی کردند. مطابق موارد قبلی، تابع درهم‌ساز h بر مقدار ورودی اعمال می‌شود و آن را به عددی از اعداد صحیح با توزیع نزدیک-یکنواخت در بازهٔ $\{۰,۱, \dots, ۲^{\ell} - ۱\}$ یا به طور معادل، به رشته‌ای از مجموعه رشته‌های دودوئی به طول ℓ تبدیل می‌کند:

$$h(x) = i = \sum_{k=۰}^{\ell-۱} ۲^k \times i_k = (i_{\ell-۱} \dots i_۱ i_۰)_۲, i_k \in \{۰,۱\} \qquad (۲)$$

فلاژوله و مارتین تشخیص دادند که احتمال حضور الگوهای $۱۰^k = ۱\overbrace{۰۰\dots۰}^{k \text{ بار}}$ در رشته‌های دودوئی برابر است و اگر برای هر مقدار نمایه شده ثبت شوند، می‌توانند نقش تخمین‌گر تعداد منحصربه‌فرد را ایفا کنند. هر الگو را می‌توان با شاخص آن، به نام «رتبه»، مرتبط کرد که از معادله زیر به دست می‌آید:

$$\text{رتبه}(i) = \begin{cases} \min_{i_k \neq ۰} k, \forall i > ۰ \\ \ell, \qquad i = ۰ \end{cases} \qquad (۳)$$

و اساسا متناظر موقعیت سمت راست‌ترین ۱ است که به عنوان ۱ با کمترین اهمیت شناخته می‌شود.

مثال ۴- محاسبهٔ رتبه

عدد صحیح ۸ بیتی ۴۲ را در نظر می‌گیریم که نمایش دودوئی آن با طرح شماره‌گذاری LSB ۰ برابر است با:

$$۴۲ = ۰ \times ۲^۷ + ۰ \times ۲^۶ + ۱ \times ۲^۵ + ۰ \times ۲^۴ + ۱ \times ۲^۳ + ۰ \times ۲^۲ + ۱ \times ۲^۱ + ۰ \times ۲^۰ = (۰۰۱۰۱۰۱۰)_۲$$

یک‌ها در موقعیت‌های ۱، ۳ و ۵ ظاهر می‌شوند. با توجه به تعریف در معادلهٔ (۳)، رتبهٔ ۴۲ برابر است با

$$\text{رتبه}(۴۲) = min\,(۱,۳,۵) = ۱$$

رخدادهای الگوی ۱۰^k، یا متناظرا $\text{رتبه}(۰) = k$، در نمایش‌های دودوئی مقادیر درهم‌ساز هر مقدار نمایه شده را می‌توان به صورت فشرده در داده‌ساختار آرایه‌ای شمارنده به طول ℓ ذخیره شوند و یا «طرح فم» نیز شناخته می‌شود.

در ابتدا، همه بیت‌ها در شمارنده برابر با صفر هستند. هنگام درج مقدار جدید x در داده‌ساختار، مقدار درهم آن را با تابع درهم‌ساز h محاسبه، سپس رتبه متناظر را محاسبه، و بیت متناظر را در آرایه برابر یک قرار می‌دهیم. مراحل در الگوریتم زیر بیان شده است.

الگوریتم ۳: درج مقدار در شمارنده ساده

ورودی: مقدار $x \in D$

ورودی: شمارنده ساده با تابع درهم‌ساز h

$j \leftarrow \text{رتبه}(h(x))$

اگر $COUNTER[j] == ۰$ آنگاه

$COUNTER[j] \leftarrow ۱$

به این ترتیب، یک در شمارنده در موقعیت j به معنای حداقل یک بار مشاهدهٔ الگوی ۱۰^j در میان مقادیر درهم شده همهٔ مقادیر نمایه شده است.

الگوریتم ۴- محاسبه رتبه x

ورودی: عدد صحیح x

خروجی: رتبه x یا کوچکترین j با شرط بیت j-ام برابر ۱

$j \leftarrow ۰$

$l = len(x)$

تازمانی که $j < l$ ادامه بده

اگر $(x \gg j \;\&\; ۱) == ۱$

برگرداندن j

$j \leftarrow j + ۱$

برگرداندن l // رخ ندادن به دلیل غیرصفر بودن h(x)

مثال ۵- ساخت شمارنده: مجموعه دادهٔ بیست نام از پایتخت‌ها اعم از برلین، برلین، پاریس، برلین، لیسبون، کیف، پاریس، لندن، رم، آتن، مادرید، وین، رم، رم، لیسبون، برلین، پاریس، لندن، کیف، واشنگتن را در نظر می‌گیریم.

از تابع درهم مورمور۳ نوع ۳۲ بیتی به عنوان تابع درهم‌ساز h استفاده می‌کنیم که مقادیر را به $\{۰,۱, \ldots, ۲^{۳۲} - ۱\}$ نگاشت می‌کند، در نتیجه از شمارنده‌ای به طول $\ell = ۳۲$ بهره می‌گیریم. با استفاده از مقادیر درهم‌ساز محاسبه شده در مثال ۳ و تعریف (۳)، رتبه‌ها را برای هر مقدار محاسبه می‌کنیم:

شهر	$h(\text{شهر})$	رتبه	شهر	$h(\text{شهر})$	رتبه
آتن	۴۱۶۱۴۹۷۸۲۰	۲	لندن	۳۴۵۰۹۲۷۴۲۲	۱
برلین	۳۶۸۰۷۹۳۹۹۱	۰	لیسبون	۶۲۹۵۵۵۲۴۷	۰
پاریس	۲۶۷۳۲۴۸۸۵۶	۳	مادرید	۲۹۷۰۱۵۴۱۴۲	۱
رم	۵۰۱۲۲۷۰۵	۰	واشنگتن	۴۰۳۹۷۴۷۹۷۹	۰
کیف	۳۴۹۱۲۹۹۶۹۳	۰	وین	۳۲۷۱۰۷۰۸۰۶	۱

بنابراین، شمارنده شکل زیر را دارد:

۱۵	۱۴	۱۳	۱۲	۱۱	۱۰	۹	۸	۷	۶	۵	۴	۳	۲	۱	۰
۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۱	۱	۱	۱

۳۱	۳۰	۲۹	۲۸	۲۷	۲۶	۲۵	۲۴	۲۳	۲۲	۲۱	۲۰	۱۹	۱۸	۱۷	۱۶
۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰

با فرض توزیع یکنواخت مقادیر در ادامه بر اساس مشاهدات نتیجه‌ای نظری را مطرح می‌کنیم. درصورتی که تعداد مقادیر متمایز تاکنون نمایه شده برابر n باشد، انتظار می‌رود که مقدار ۱ در موقعیت اول در حدود $\frac{n}{۲}$ دفعه ظاهر شود، ظاهر شدن مقدار یک در موقعیت دوم در حدود $\frac{n}{۲^۲}$ دفعات، و به همین ترتیب تعداد رخدادها در هر موقعیت به دست م‌آید. تعداد ظهور یک در موقعیت k برابر $\frac{n}{۲^k}$ است. اگر $j \gg log_۲ n$ باشد، احتمال مشاهدهٔ مقدار ۱ در موقعیت j-ام به صفر میل می‌کند. در نتیجه مقدار Counter[j]

تقریبا با قطعیت صفر خواهد بود. به طور مشابه، اگر $j \ll log_2 n$ باشد، مقدار Counter[j] تقریبا با قطعیت برابر با یک خواهد بود. اگر مقدار j در حدود $log_2 n$ باشد، احتمال مشاهدهٔ ۱ یا ۰ در آن موقعیت تقریبا برابر است.

در نتیجه، پس از درج تمام مقادیر مجموعه‌داده در آرایهٔ شمارنده، می‌توان سمت راست‌ترین موقعیتی (R) را که مقدار صفر ظاهر می‌شود به عنوان شاخص تخمین $log_2 n$ در نظر گرفت. فلاژوله و مارتین نشان دادند که در عمل، ضریب تصحیح ϕ جهت بهبود تخمین لازم است و تخمین کاردینالیتی (تعداد مقادیر متمایز) با معادلهٔ زیر انجام می‌شود:

$$n \approx \frac{1}{\phi} 2^R \qquad (۴)$$

که $\phi \approx ۰.۷۷۳۵۱$ است. سخن کوتاه، فلاژوله و مارتین از موقعیت کم‌ارزش‌ترین بیت صفر (راست‌ترین صفر) به عنوان تخمین کاردینالیتی استفاده کردند و الگوریتم خود را بر همین اساس ساختند. با این حال، از مشاهدهٔ بالا می‌توان دید که موقعیت پرارزش‌ترین بیت ۱ (چپ‌ترین ۱) نیز می‌تواند به همین منظور استفاده شود. اما توزیع آن یکنواخت‌تر است و در نتیجه خطای استاندارد بزرگ‌تری ایجاد می‌کند. الگوریتم محاسبهٔ راست‌ترین موقعیت صفر در شمارندهٔ ساده را می‌توان به صورت زیر معادله‌بندی کرد.

الگوریتم ۵- محاسبه موقعیت سمت چپ‌ترین صفر
ورودی: شمارنده ساده به طول ℓ
خروجی: سمت چپ‌ترین موقعیت صفر
برای j از ۰ تا ۱ − ℓ انجام بده
اگر ۰ == COUNTER[j] آن‌گاه
برگرداندن j
برگرداندن ℓ

مثال ۶: تخمین تعداد منحصربه‌فرد با شمارنده ساده
آرایهٔ مثال ۵ را در نظر می‌گیریم و تعداد تخمینی مقادیر منحصربه‌فرد را محاسبه کنید.

۰	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱	۱۲	۱۳	۱۴	۱۵
۱	۱	۱	۱	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰

۱۶	۱۷	۱۸	۱۹	۲۰	۲۱	۲۲	۲۳	۲۴	۲۵	۲۶	۲۷	۲۸	۲۹	۳۰	۳۱
۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰

با استفاده از الگوریتم ۵، در COUNTER چپ‌ترین مقدار ۰ در موقعیت R = ٤ ظاهر می‌شود، بنابراین، با توجه به معادله (۴)، تخمین تعداد منحصربه‌فرد است

$$n \approx \frac{1}{۰.۷۷۳۵۱} 2^4 \approx ۲۰.۶۸$$

تعداد دقیق منحصربه‌فرد مجموعه ۱۰ است، در نتیجه حاصل تخمین دارای خطای زیادی است زیرا مقادیر R اعداد صحیح هستند و برای رتبه‌های بسیار نزدیک می‌توانیم نتایجی را به دست آوریم که دارای تفاوتی از مرتبه توان‌های دودوئی باشند. در مثال حاضر نیز، R=۳ تخمین نسبتا مناسب ۱۰.۳۴ را تولید می‌کند.

تخمین تعداد منحصربه‌فرد بر اساس شمارنده معمول می‌تواند مقادیر مورد انتظار بسیار نزدیکی را ارائه دهد، اما دارای وردایی نسبتا بالایی است که معمولا، همانطور که در مثال قبل نیز مشاهده کردیم، با خطای استاندارد δ دارای اختلاف از مرتبه دو برابر است.

ناگفته پیداست، ضعف رویکرد تک‌شمارنده فقدان تخمین‌های با اطمینان بالا برای تعداد منحصربه‌فرد است. در واقع، پیش‌بینی خود را تنها بر اساس تخمینی واحد انجام می‌دهد. از این رو، بهبود منطقی الگوریتم استفاده از تعداد زیادی شمارنده و در نتیجه افزایش تعداد تخمین‌ها است. پیش‌بینی نهایی n را می‌توان با میانگین‌گیری از پیش‌بینی‌های R_i از شمارنده‌های مزبور $\{\text{شمارنده}_i\}_{i=0}^{m-1}$ به دست آورد. سخن کوتاه، معادله اصلاح شده (۴) الگوریتم شمارش احتمالاتی به شکل زیر است:

$$n \approx \frac{1}{\phi} 2^{\frac{1}{m}\sum_{i=0}^{m-1} R_i} \qquad (۵)$$

و تعداد منحصربه‌فرد n مقدار تخمینی با همان کیفیت اما با وردایی بسیار کوچکتر را در پی خواهد داشت. اشکال عملی برای ساخت m شمارنده مستقل، نیاز به محاسبه مقادیر m تابع درهم‌ساز متفاوت است که با توجه به اینکه هر تابع درهم‌ساز را می‌توان در $O(1)$ محاسبه کرد، محاسبه همهٔ آنها دارای پیچیدگی زمانی $O(m)$ و در نتیجه هزینه‌های پردازشی نسبتا بالا را در پی دارد. راه‌حل بهینه‌سازی الگوریتم شمارش احتمالاتی، اعمال «میانگین‌گیری تصادفی» است. به دیگر سخن، m تابع درهم‌ساز صرفا با یک تابع جانشین می‌شوند اما حاصل مقدار آن به دو بخش خارج‌قسمت و باقیمانده تقسیم می‌شود. باقیمانده r برای انتخاب یکی از m شمارنده و خارج‌قسمت q برای محاسبه رتبه و یافتن شاخص مناسب برای به‌روزرسانی در آن شمارنده استفاده می‌شود.

الگوریتم ۶: استفاده از میانگین‌گیری برای به‌روزرسانی شمارنده‌ها
ورودی: مقدار $x \in D$
ورودی: آرایه‌ای از m شمارنده ساده با تابع درهم‌ساز h
$f_r \leftarrow h(x) \% m$
$f_q \leftarrow \left\lfloor \frac{h(x)}{m} \right\rfloor$
$j \leftarrow rank(f_q)$
$COUNTER_{f_r}[j] \leftarrow 1$

با اعمال الگوریتم میانگین‌گیری تصادفی ۶ به شمارش احتمالاتی، با این فرض که توزیع مقادیر مبتنی بر خارج‌قسمت به اندازه کافی منصفانه است، ممکن است انتظار داشته باشیم که $\frac{n}{m}$ مقدار توسط هر شمارنده ساده $\{COUNTER_i\}_{i=0}^{m-1}$ اندیس شده باشند، بنابراین معادله (۵) تخمین خوبی برای $\frac{n}{m}$ و نه مستقیما n است. پس معادله‌ای که فلاژوله و مارتین پیشنهاد دادند به صورت زیر و الگوریتم پیشنهادی آنها به صورت زیر است:

$$n \approx \frac{m}{\phi} 2^{\bar{R}} = \frac{m}{\phi} 2^{\frac{1}{m}\sum_{i=0}^{m-1} R_i} \qquad (۶)$$

الگوریتم ۷: الگوریتم فلاژوله-مارتین (PCSA)
ورودی: مجموعه داده D
ورودی: آرایه‌ای از m شمارنده ساده با تابع درهم‌ساز h
خروجی: تخمین تعداد منحصربه‌فرد
برای $x \in D$ انجام بده
$f_r \leftarrow h(x) \% m$
$f_q \leftarrow \left\lfloor \frac{h(x)}{m} \right\rfloor$
$j \leftarrow rank(f_q)$

$COUNTER_{f_r}[j] \leftarrow ۱$
برای r از ۰ تا $m - ۱$ انجام بده
$R \leftarrow LeftMostZero(COUNTER_{f_r})$
$S \leftarrow S + R$
$S \leftarrow \frac{۱}{\mathrm{m}} S$
برگرداندن $\frac{m}{\phi}\ ۲^S$

الگوریتم بالا الگوریتم شمارش احتمالاتی با میانگین‌گیری تصادفی (PCSA) نامیده می‌شود و همچنین الگوریتم فلاژوله-مارتین نیز شناخته می‌شود. در مقایسه با نسخه‌ای که از m تابع درهم‌ساز استفاده می‌کند، پیچیدگی زمانی را برای هر مقدار به حدود $O(۱)$ کاهش می‌دهد.

مثال۷- تخمین تعداد منحصربه‌فرد با میانگین‌گیری تصادفی: مجموعه داده و مقادیر درهم‌ساز محاسبه شده در مثال ۶ را در نظر می‌گیریم و فن میانگین‌گیری تصادفی را با شبیه‌سازی $m = ۳$ تابع درهم‌ساز اعمال می‌کنیم. از باقیمانده r برای انتخاب یکی از سه شمارنده و از خارج‌قسمت q برای محاسبه رتبه استفاده می‌کنیم.

شهر	$h(\text{شهر})$	r	Q	رتبه(q)	شهر	$h(\text{شهر})$	r	q	رتبه(q)
آتن	۴۱۶۱۴۹۷۸۲۰	۰	۱۳۸۷۱۶۵۹۴۰	۲	لندن	۳۴۵۰۹۲۷۴۲۲	۲	۱۱۵۰۳۰۹۱۴۰	۲
برلین	۳۶۸۰۷۹۳۹۹۱	۱	۱۲۲۶۹۳۱۳۳۰	۱	لیسبون	۶۲۹۵۵۵۲۴۷	۰	۲۰۹۸۵۱۷۴۹	۰
پاریس	۲۶۷۳۲۴۸۸۵۶	۰	۸۹۱۰۸۲۹۵۲	۳	مادرید	۲۹۷۰۱۵۴۱۴۲	۲	۹۹۰۰۵۱۳۸۰	۲
رم	۵۰۱۲۲۷۰۵	۱	۱۶۷۰۷۵۶۸	۴	واشنگتن	۴۰۳۹۷۴۷۹۷۹	۲	۱۳۴۶۵۸۲۶۵۹	۰
کیف	۳۴۹۱۲۹۹۶۹۳	۱	۱۱۶۳۷۶۶۵۶۴	۲	وین	۳۲۷۱۰۷۰۸۰۶	۱	۱۰۹۰۳۵۶۹۳۵	۰

هر شمارنده اطلاعات مربوط به حدود یک سوم شهرها را مدیریت می‌کند، بنابراین، توزیع به اندازه کافی منصفانه است. پس از نمایه کردن همه مقادیر و تنظیم بیت‌های مناسب در شمارنده‌های مربوطه، شمارنده‌های حاصل به دست می‌آیند. شمارنده ۰ شکل زیر را دارد:

۰	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱	۱۲	۱۳	۱۴	۱۵
۱	۰	۱	۱	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰

۱۶	۱۷	۱۸	۱۹	۲۰	۲۱	۲۲	۲۳	۲۴	۲۵	۲۶	۲۷	۲۸	۲۹	۳۰	۳۱
۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰

شمارنده ۱ به صورت زیر است:

۰	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱	۱۲	۱۳	۱۴	۱۵
۱	۱	۱	۰	۱	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰

۱۶	۱۷	۱۸	۱۹	۲۰	۲۱	۲۲	۲۳	۲۴	۲۵	۲۶	۲۷	۲۸	۲۹	۳۰	۳۱
۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰

شمارنده ۲ به صورت زیر است:

۰	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱	۱۲	۱۳	۱۴	۱۵
۱	۰	۱	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰

۱۶	۱۷	۱۸	۱۹	۲۰	۲۱	۲۲	۲۳	۲۴	۲۵	۲۶	۲۷	۲۸	۲۹	۳۰	۳۱
۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰

سمت چپ‌ترین موقعیت‌های صفر برای هر شمارنده (که در بالا مشخص شده‌اند) عبارت از $R_{۰} = ۱$، $R_{۱} = ۳$ و $R_{۲} = ۱$ هستند. بنابراین، تخمین تعداد منحصربه‌فرد با توجه به معادله (۶) است

$$n \approx \frac{۳}{\phi}\, ۲^{\frac{۱}{۳}\sum_{k=۰}^{۲} R_k} \approx \frac{۳}{۰.۷۷۳۵۱}\, ۲^{\frac{۱+۳+۱}{۳}} \approx ۱۲.۳۱$$

تخمین حاصل بسیار نزدیک‌تر به مقدار واقعی تعداد منحصربه‌فرد ۱۰ شده است، و حتی بدون استفاده از شمارنده‌های زیاد، به طور قابل توجهی از تخمین مثال ۶ بهتر عمل می‌کند.

ویژگی‌ها

الگوریتم فلاژوله-مارتین برای مجموعه‌های داده با تعداد منحصربه‌فرد زیاد به خوبی کار می‌کند و در نسبت‌های $\frac{n}{m} > ۲۰$ تقریب‌های خوبی به دست می‌دهد. با این حال، می‌توان از الگوریتم‌های غیرخطی در الگوریتم برای کار با مجموعه داده‌های با تعداد منحصربه‌فردهای کوچک بهره برد تا نتایج را بهبود داد. بیورن شویرمان و مارتین ماوه در سال ۱۳۸۶ اصلاحی احتمالی پیشنهاد کردند که و معادلهٔ (۶) را با افزودن عبارتی برای تعداد منحصربه‌فردهای کوچک متناسب ساختند که برای تعداد منحصربه‌فردهای بزرگ به صفر همگرا می‌شود:

$$n \approx \frac{m}{\phi}\left(۲^{\bar{R}} - ۲^{-\chi\bar{R}}\right) \qquad (۷)$$

که $\chi \approx ۱.۷۵$. خطای استاندارد δ الگوریتم فلاژوله-مارتین با تعداد شمارنده‌های استفاده شده رابطه معکوس دارد و می‌توان آن را به صورت تقریبی محاسبه کرد:

$$\delta \approx \frac{۰.۷۸}{\sqrt{m}}$$

مقادیر مرجع خطای استاندارد برای تعداد شمارنده‌های پرکاربرد را می‌توان در جدول ۵ یافت. طول ℓ هر شمارنده را می‌توان به گونه‌ای انتخاب کرد که:

$$M > log_{۲}\left(\frac{n}{m}\right) + ۴$$

بنابراین، $\ell = ۳۲$ عملا برای شمارش تعداد منحصربه‌فردهای بسیار فراتر از $۱۰^{۹}$ با استفاده از ۶۴ شمارنده کافی است.

جدول ۵- تعادل بین دقت و ذخیره‌سازی ($\ell = ۳۲$)

m	حافظه	δ
۶۴	۲۵۶ بایت	۹.۷ درصد
۲۵۶	۱.۰۲۴ کیلوبایت	۴.۸ درصد
۱۰۲۴	۴.۱ کیلوبایت	۲.۴ درصد

شمارنده‌های متناظر مجموعه‌های داده مختلف را می‌توان به راحتی با اعمال عملیات OR بیتی با هم ادغام کرد که نتیجه آن یک شمارنده برای کل آن مجموعه داده‌هاست. مانند فیلتر بلوم، الگوریتم‌های شمارش احتمالاتی از حذف پشتیبانی نمی‌کنند. اما، از رویکردی مشابه بلوم شمارنده و با افزایش فضا می‌توان حذف را پوشش داد.

روش‌های لگ‌لگ و ابرلگ‌لگ

پرکاربردترین الگوریتم‌های احتمالاتی برای تخمین تعداد منحصربه‌فرد که در عمل استفاده می‌شوند، خانواده لگ‌لگ از الگوریتم‌ها است که شامل الگوریتم لگ‌لگ LogLog است که پیشنهادی ماریان دوراند و فیلیپ فلاژوله در سال ۱۳۸۲، و دو روش جانشین آن ابرلگ‌لگ و ابرلگ‌لگ++ است. الگوریتم‌ها از رویکردی مشابه الگوریتم شمارش احتمالاتی استفاده می‌کنند. به بیان دیگر، تخمین تعداد منحصربه‌فرد n با مشاهده حداکثر تعداد صفرهای ابتدایی در نمایش دودوئی مقادیر انجام می‌شود. همه آنها به حافظه کمکی نیاز دارند و داده‌ها را یکبار پیمایش می‌کنند تا تخمینی از تعداد منحصربه‌فرد تولید کنند.

مطابق منوال روش‌های قبلی تعیین تعداد منحصر بفرد اجزا، هر مقدار در مجموعه داده با اعمال تابع درهم‌سازی پیش‌پردازش می‌شود که مقادیر را به اعداد صحیح با توزیع کافی یکنواخت در بازهٔ $\{۰,۱, \dots, ۲^{\ell-۱}\}$ یا به طور معادل، بر روی مجموعه رشته‌های دودوئی به طول M تبدیل می‌کند:

$$h(x) = i = \sum_{k=۰}^{\ell-۱} i_k ۲^k = (i_{\ell-۱} \dots i_۱ i_۰)_۲, i_k \in \{۰,۱\}$$

مراحل الگوریتم‌ها مشابه PCSA است که در اینجا بار دیگر یادآوری می‌کنیم. ابتدا، مجموعه دادهٔ اولیه یا جریان ورودی را به تعدادی زیرمجموعه تقسیم می‌کند که هر کدام با یکی از m شمارنده ساده نمایه می‌شوند. سپس، با توجه به میانگین‌گیری تصادفی، زیرا تابع درهم‌ساز واحد وجود دارد، شمارنده را برای مقدار خاص x با استفاده از یک بخش از مقدار درهم h(x) انتخاب می‌کنیم، در حالی که بخش دیگر برای به‌روزرسانی شمارنده مربوطه استفاده می‌شود.

همه الگوریتم‌های مورد بحث در اینجا بر اساس مشاهده الگوهای ۱۰^k هستند که در ابتدای مقادیر برای شمارنده خاص رخ می‌دهند، و هر الگو را با شاخص آن، به نام رتبه، مرتبط می‌کنند. رتبه معادل موقعیت ۱ با کمترین اهمیت در نمایش دودوئی مقدار درهم‌ساز مقدار نمایه شده است و می‌تواند با معادله (۳) محاسبه شود. هر شمارنده مشاهده تعداد منحصربه‌فرد خود را بر اساس رتبه‌های دیده شده می‌سازد، و تخمین نهایی تعداد منحصربه‌فرد از چنین مشاهداتی با استفاده از تابع ارزیابی تولید می‌شود.

در مورد ذخیره‌سازی، نگهداری شمارنده‌ها در الگوریتم شمارش احتمالاتی نسبتا پرهزینه است، اما الگوریتم لگ‌لگ راه‌حلی فضا-کارآمدتر را همراه با تابع ارزیابی بهتر و رویکرد تصحیح سوگیری پیشنهاد می‌کند.

الگوریتم لگ‌لگ

الگوریتم لگ‌لگ بر اساس محاسبهٔ رتبهٔ هر مقدار ورودی با تابع درهم‌ساز واحد h قرار دارد. در صورتی که n تعداد کل عنصرهای نمایه شده در شمارنده است، می‌توان ادعا کرد که $\frac{n}{۲^k}$ مقدار رتبهٔ k اختیار کنند پس رتبهٔ بیشینه مشاهده شده نشانه و تقریب مناسبی از مقدار $\log_۲ n$ فراهم کند:

$$R = max_{x \in D}\big(rank(x)\big) \approx log_۲ n \qquad (۸)$$

ولی تخمین مذکور دارای خطایی با مرتبهٔ ۱.۸۷ برابر مقدار واقعی بود که عملا بی‌معنی است. برای کاهش خطا، الگوریتم لگ‌لگ از سطل‌بندی مبتنی بر میانگین‌گیری تصادفی استفاده می‌کند و مجموعه داده را به $m = ۲^p$ زیرمجموعه $\{S_۰, S_۱, \dots, S_{m-۱}\}$ تقسیم می‌کند، که در آن پارامتر دقت p تعداد بیت‌های استفاده شده را تعریف می‌کند. بنابراین، برای هر مقدار x از مجموعه داده، p بیت اول مقدار درهم ℓ -بیتی h(x) را می‌توان برای یافتن شاخص j زیرمجموعه مناسب گرفت:

$$j = \left(i_{p-۱} \dots i_۱ i_۰ \right)_۲$$

و بقیه (ℓ - p) بیت در شمارنده متناظر COUNTER[j] اندیس می‌شوند تا رتبه محاسبه شده و مشاهده R_j طبق معادله (۸) به دست آید.

با توزیع منصفانه، هر زیرمجموعه دارای n/m عضو خواهد بود. پس، R_j-ها در شمارنده‌ها $\{COUNTER[j]\}_{j=۰}^{m-۱}$ می‌توانند نشانه‌ای از مقدار $\log_۲ \frac{n}{m}$ ارائه دهند، و با استفاده از میانگین حسابی آنها و تصحیح سوگیری، می‌توان وردائی مشاهده را کاهش داد:

$$n = \alpha_m \cdot m \cdot ۲^{\frac{۱}{m} \sum_{j=۰}^{m-۱} R_j}$$

که در آن $\alpha_m = \left(\Gamma\left(-\frac{۱}{m}\right) \times \frac{۱ - ۲^{\frac{۱}{m}}}{\log ۲} \right)^m \approx ۰.۳۹۷۰۱ - \frac{۲\pi^۲ + (ln ۲)^۲}{۴۸m}$، و $\Gamma(۰)$ تابع گاما است. با این حال، برای اکثر موارد عملی $m \geq ۶۴$ استفاده از $\alpha_m \approx ۰.۳۹۷۰۱$ کفایت می‌کند.

الگوریتم ۸- تخمین تعداد منحصربه‌فرد با لگ‌لگ

ورودی: مجموعه داده D

ورودی: آرایه‌ای از m شمارنده لگ‌لگ با تابع درهم‌ساز h

خروجی: تخمین تعداد منحصربه‌فرد

$COUNTER[j] \leftarrow ۰, j = ۰ \dots m - ۱$

برای $x \in D$ انجام بده

$i \leftarrow h(x) = (i_{\ell-۱} \dots i_۱ i_۰)_۲, i_k \in \{۰,۱\}$

$j \leftarrow (i_{p-۱} \dots i_۱ i_۰)_۲$

$r \leftarrow rank((i_{\ell-۱} \dots i_{p+۱} i_p)_۲)$

$COUNTER[j] \leftarrow max(COUNTER[j], r)$

$R \leftarrow \frac{۱}{m} \sum_{j=۰}^{m-۱} Counter[j]$

برگرداندن $\alpha_m \cdot m \cdot ۲^R$

مثال ۸- فرض کنید m=۴ پس در نتیجه p = log۴=۲ بیت جهت انتخاب شمارنده نیاز داریم. فرض می‌کنیم خروجی درهم طولی برابر هشت بیت دارد. پس بیت‌های صفر و یک برای شمارنده و بقیه شش بیت برای محاسبهٔ رتبه استفاده می‌شود.

شهر	h(x) (۸ bits)	($b_۱b_۰$)	($b_۷b_۶b_۵b_۴b_۳b_۲$)
Tehran	۰۱۱۰۱۰۱۰	۱۰ = ۲	**۰۱۱۰۱۰**
Paris	۱۰۱۱۰۰۱۱	۱۱ = ۳	**۱۰۱۱۰۰**
London	۰۰۰۱۱۱۰۰	۰۰ = ۰	**۰۰۰۱۱۱**
Berlin	۱۱۰۰۱۰۱۰	۱۰ = ۲	**۱۱۰۰۱۰**

Rome	۰۰۱۰۱۱۰۱	۰۱ = ۱	**۰۰۱۰۱۱**
Madrid	۰۱۰۰۱۱۱۰	۱۰ = ۲	**۰۱۰۰۱۱**
Kyiv	۱۰۰۰۰۱۱۱	۱۱ = ۳	**۱۰۰۰۰۱**
Vienna	۱۱۱۰۰۰۰۰	۰۰ = ۰	**۱۱۱۰۰۰**

در قدم بعدی رتبه بیت‌های بالایی حساب می‌شود.

شهر	$(b_7b_6b_5b_4b_3b_2)$	رتبه
Tehran	۰۱۱۰۱۰	**۱**
Paris	۱۰۱۱۰۰	**۲**
London	۰۰۰۱۱۱	**۰**
Berlin	۱۱۰۰۱۰	**۱**
Rome	۰۰۱۰۱۱	**۰**
Madrid	۰۱۰۰۱۱	**۰**
Kyiv	۱۰۰۰۰۱	**۰**
Vienna	۱۱۱۰۰۰	**۳**

در قدم بعدی شمارنده‌ها با عملیات بیش بروز می‌شوند.

شمارنده j	شهرهای تخصیص یافته	رتبه	بیش شمارنده
۰	لندن با r=۰ و وین با r=۳	{۰,۳}	**۳**
۱	رم با r=۰	{۰}	**۰**
۲	طهران با r=۱ و برلین با r=۱ و مادرید با r=۰	{۰,۱,۱}	**۱**
۳	پاریس با r=۲ و کیف با r=۰	{۰,۲}	**۲**

پس شمارنده دارای مقدار [۳,۰,۱,۲] است. میانگین را به صورت زیر حساب می‌کنیم.

$$R = \frac{۱}{m}\sum \text{شمارنده}[j] = \frac{۱}{۴}(۳ + ۰ + ۱ + ۲) = ۱.۵$$

نتیجه نهایی تخمین برای m=٤ و $\alpha_۴ \approx ۰.۶۷$ برابر است با

$$\hat{n} = \alpha_m \times m \times ۲^R = ۰.۶۷ \times ۴ \times ۲^{۱.۵} = ۷.۵۸$$

تعداد واقعی برابر با ۸ است و تخمین لگ‌لگ برابر ۷.۶ است و خطا در حدود پنج درصد است.

ویژگی‌ها

خطای استاندارد δ الگوریتم لگ‌لگ با تعداد شمارنده‌های استفاده شده m رابطه معکوس دارد و می‌توان آن را به صورت تقریبی محاسبه کرد:

$$\delta \approx \frac{۱.۳}{\sqrt{m}}$$

بنابراین، برای m = ۲٥٦ خطای استاندارد حدود هشت درصد و برای m = ۱۰۲٤ به حدود چهار درصد کاهش می‌یابد.

اگر شمارش تا n لازم باشد، فضای مورد نیاز ذخیره‌سازی الگوریتم لگ‌لگ تخمینی از $O(\log_۲ \log_۲ n)$ است. به طور دقیق‌تر، کل فضای مورد نیاز الگوریتم برای شمارش تا n برابر $m.\log_۲ \log_۲ \frac{n}{m}\big(۱ + O(۱)\big)$ است.

در مقایسه با الگوریتم شمارش احتمالاتی که در آن هر شمارنده به ۱۶ یا ۳۲ بیت نیاز دارد، الگوریتم لگ‌لگ به شمارنده‌های بسیار کوچکتر $\{COUNTER[j]\}_{j=0}^{m-1}$ معمولا هر کدام در حدود پنج بیت نیاز دارد. با وجود مصرف حافظهٔ کمتر الگوریتم لگ لگ نسبت به الگوریتم شمارش احتمالاتی، به میزان ناچیزی دقت کاهش می‌یابد.

در صورتی که بهشمارش تعداد منحصربه‌فرد تا حدود 2^{30}، یعنی حدود ۱ میلیارد، با دقت حدود چهار درصد نیاز باشد. همانطور که گفته شد، برای چنین خطای استانداردی، $m = 1024$ سطل مورد نیاز است که هر کدام حدود $\frac{n}{m} = 2^{20}$ مقدار دریافت می‌کنند. تعداد بیت موردنیاز $\log_2 \log_2 2^{20} \approx 4.32$، یا حدود ۵ بیت به ازای هر سطل (مقداری کمتر از ۳۲) کافی است. بنابراین، برای تخمین تعداد منحصربه‌فرد تا حدود 10^9 با خطای استاندارد چهار درصد، الگوریتم به ۱۰۲۴ سطل ۵ بیتی نیاز دارد که در مجموع ۶۴۰ بایت است.

الگوریتم ابرلگ‌لگ

فیلیپ فلاژوله، اریک فیوزی، اولیویه گاندوئه، و فردریک مونیه در سال ۱۳۸۶ الگوریتم ابرلگ‌لگ را بر اساس الگوریتم لگ‌لگ پیشنهاد دادند. الگوریتم ابرلگ‌لگ از تابع درهم‌ساز ۳۲ بیتی، تابع ارزیابی متفاوت و تصحیحات سوگیری مختلف استفاده می‌کند. مشابه الگوریتم لگ‌لگ، ابرلگ‌لگ از تصادفی‌سازی برای تقریب تعداد منحصربه‌فرد مجموعه داده استفاده می‌کند و برای مدیریت تعداد منحصربه‌فرد تا 10^9 با یک تابع درهم‌ساز ۳۲ بیتی h طراحی شده است و مجموعه داده را به $m = 2^p$ زیرمجموعه، با دقت $p \in 4 \dots 16$ تقسیم می‌کند.

علاوه بر این، تابع ارزیابی الگوریتم ابرلگ‌لگ را از لگ‌لگ استاندارد متمایز می‌کند. الگوریتم لگ‌لگ اصلی از میانگین هندسی استفاده می‌کند در حالی که ابرلگ‌لگ از تابعی استفاده می‌کند که مبتنی بر نسخه نرمال شده میانگین هارمونیک است:

$$\hat{n} \approx \alpha_m \cdot m^2 \cdot \left(\sum_{j=0}^{m-1} 2^{-Coutner[j]} \right)^{-1} \qquad (9)$$

که در آن

$$\alpha_m = \left(m \int_0^\infty \left(\log_2 \left(\frac{2+x}{1+x} \right) \right)^m dx \right)^{-1} = \frac{1}{2ln2 \left(1 + \frac{1}{m}(3ln2 - 1) + O(m^{-2}) \right)}$$

مقادیر تقریبی α_m را در جدول زیر یافت. شهود استفاده از میانگین هارمونیک این است که وردائی را به دلیل ویژگی آن در هموارسازی توزیع‌های احتمال چوله کاهش می‌دهد.

جدول ۶- α_m برای پرکاربردترین مقادیر m

m	α_m
2^4	۰.۶۷۳
2^5	۰.۶۹۷
2^6	۰.۷۰۹
$\geq 2^7$	$\frac{0.7213 \times m}{m + 1.079}$

تخمین (۹) به دلیل خطاهای غیرخطی نیاز به تصحیح برای محدوده‌های کوچک و بزرگ دارد. فلاژوله و همکاران در نتیجهٔ آزمایش تجربی دریافتند که برای تعداد منحصربه‌فردهای کوچک $n < \frac{۵}{۲}m$ برای دستیابی به تخمین‌های بهتر، الگوریتم ابرلگ‌لگ را می‌توان با شمارش خطی با استفاده از تعدادی از شمارنده‌های COUNTER[j] غیرصفر تصحیح کرد (اگر یک شمارنده مقدار صفر داشته باشد، می‌توانیم با قطعیت بگوییم که آن زیرمجموعه خاص خالی است). بنابراین، برای محدوده‌های مختلف تعداد منحصربه‌فرد، که به صورت فواصل روی تخمین $\hat{n}$ محاسبه شده توسط معادله (۳.۱۲) بیان می‌شوند، الگوریتم تصحیحات زیر را ارائه می‌دهد:

$$n = \begin{cases} LinearCounter, \hat{n} \leq \frac{۵}{۲}m \wedge \exists j : Counter[j] \neq ۰ \\ -۲^{۳۲} \log\left(۱ - \frac{\hat{n}}{۲^{۳۲}}\right), \quad \hat{n} > \frac{۱}{۳۰}۲^{۳۲} \\ \hat{n}, \quad \text{دغا} \end{cases}$$

با این حال، برای n = ۰ تصحیح به درستی کار نمی‌کند و الگوریتم همیشه تقریبا $۰.۷m$ را برمی‌گرداند.

به دلیل استفاده الگوریتم ابرلگ‌لگ از تابع درهم‌ساز ۳۲ بیتی، هنگامی که تعداد منحصربه‌فرد به $۱۰^{۹} \times ۴ \approx ۲^{۳۲}$ نزدیک می‌شود، تابع درهم‌ساز تقریبا به حد خود می‌رسد و احتمال تصادم افزایش می‌یابد. برای چنین محدوده‌های بزرگ، الگوریتم ابرلگ‌لگ تعداد مقادیر درهم‌ساز مختلف را تخمین می‌زند و از آن برای تقریب تعداد منحصربه‌فرد استفاده می‌کند. با این حال، در عمل، این خطر وجود دارد که تعداد بالاتر به سادگی قابل نمایش نباشد و از بین برود و بر دقت تأثیر بگذارد.

تابع درهم‌ساز را در نظر می‌گیریم که دامنه را به مقادیر M بیت نگاشت می‌کند. چنین تابعی حداکثر می‌تواند ۲^{M} مقدار مختلف را رمزگذاری کند و اگر تعداد منحصربه‌فرد تخمینی n به چنین حدی نزدیک شود، تصادم‌های درهم‌ساز محتمل‌تر می‌شوند.

هیچ نشانی دال بر عملکرد بهتر توابع درهم‌ساز پرکاربرد (مانند MurmurHash۳، MD۵، SHA-۱، SHA-۲۵۶) نسبت به سایر روش‌ها در الگوریتم‌های ابرلگ‌لگ یا اصلاحات آن وجود ندارد.

الگوریتم کامل ابرلگ‌لگ در زیر نشان داده شده است.

الگوریتم ۹- تخمین تعداد منحصربه‌فرد با ابرلگ‌لگ
ورودی: مجموعه داده D
ورودی: آرایه‌ای از m شمارنده LogLog با تابع درهم‌ساز h
خروجی: تخمین تعداد منحصربه‌فرد
$COUNTER[j] \leftarrow ۰, j = ۰ \dots m - ۱$
برای $x \in D$ انجام بده
$i \leftarrow h(x) = (i_{۳۱} \dots i_{۱} i_{۰})_{۲}$
$j \leftarrow (i_{p-۱} \dots i_{۱} i_{۰})_{۲}$
$r \leftarrow rank((i_{۳۱} \dots i_{p+۱} i_{p})_{۲})$
$COUNTER[j] \leftarrow max(COUNTER[j], r)$
$R \leftarrow \sum_{j=۰}^{m-۱} ۲^{-Counter[j]}$
$\hat{n} = \alpha_m \cdot m^{۲} \cdot \frac{۱}{R}$
$n \leftarrow \hat{n}$

اگر $\hat{n} \le \frac{5}{2}m$ آنگاه
$Z \leftarrow$ تعداد$_{j=0\ldots m-1}(COUNTER[j] = 0)$
اگر $Z \neq 0$ آنگاه
$$n \leftarrow m \cdot log\left(\frac{m}{Z}\right)$$
در غیر این صورت اگر $\hat{n} > \frac{1}{30}2^{32}$ آنگاه
$$n \leftarrow -2^{32} \cdot \log\left(1 - \frac{n}{2^{32}}\right)$$
برگرداندن n

مثال۹- در مثال قبل مقدار شمارنده‌ها دارای مقدار [۳،۰،۱،۲] است. میانگین را به صورت زیر حساب می‌کنیم.
$$R = \sum 2^{-\text{شمارنده}[j]} = (2^{-3} + 2^{0} + 2^{-1} + 2^{-2}) = \frac{1}{8} + 1 + \frac{1}{2} + \frac{1}{4} = \frac{15}{8} = 1.875$$
نتیجه نهایی تخمین برای m=4 و $\alpha_4 \approx 0.673$ برابر است با
$$\hat{n} = \alpha_m \times m^2 \times \frac{1}{R} = 0.673 \times 16 \times \frac{1}{1.875} = 5.74$$
تعداد واقعی برابر با ۸ است و تخمین لگ‌لگ برابر ۵.۷۴ است.

ویژگی‌ها

مشابه الگوریتم لگ‌لگ، بین خطای استاندارد δ و تعداد شمارنده‌ها m نیاز به سبک-سنگینی وجود دارد:

$$\delta \approx \frac{1.04}{\sqrt{m}}$$

نیاز حافظه با تعداد مقادیر به صورت خطی رشد نمی‌کند (برخلاف الگوریتم‌هائی مانند الگوریتم شمارش خطی)، با تخصیص (M-p) بیت برای مقادیر درهم‌ساز و داشتن $m = 2^p$ شمارنده در مجموع، حافظه مورد نیاز برابر است با
$$\lceil \log_2(M + 1 - p) \rceil \cdot 2^p \; bits$$

به بیان دیگر، هر شمارنده به $ceil(log_2($رتبه حداکثر$ + 1))$ بیت نیاز دارد. علاوه بر این، الگوریتم فقط از توابع درهم‌ساز ۳۲ بیتی و دقت $p \in 4 \ldots 16$ استفاده می‌کند، نیازهای حافظه برای داده‌ساختار ابرلگ‌لگ برابر با $2^p \times 5$ بیت است.

بنابراین، الگوریتم ابرلگ‌لگ تخمین تعداد منحصربه‌فردهای بسیار فراتر از 10^9 را با دقت دو درصد و با استفاده از تنها ۱.۵ کیلوبایت حافظه ممکن می‌سازد.

به عنوان مثال، پایگاه داده معروف درون‌حافظه Redis ساختارهای داده ابرلگ‌لگ با ۱۲ کیلوبایت را حفظ می‌کند که تعداد منحصربه‌فردها را با خطای استاندارد ۰.۸۱ درصد تقریب می‌زند.جدول زیر مقایسه ای از ابرلگ لگ را با دیگر الگوریتم‌ها نشان می‌دهد.

روش	حافظه مور نیاز برای یک میلیارد مقدار یکتا	خطا
شمارش دقیق	حدود ۳۲ گیگابایت	۰ درصد
شمارش خطی	حدود یک گیگابایت	۱ درصد
ابرلگ‌لگ	حدود ۱۲ کیلوبایت	۰.۸۱ درصد

در حالی که ابرلگ‌لگ در مقایسه با لگ‌لگ تخمین تعداد منحصربه‌فرد را برای مجموعه‌های داده کوچک بهبود بخشید، هنوز هم تعداد منحصربه‌فردهای واقعی را در چنین مواردی بیش از حد تخمین می‌زند. انواع الگوریتم‌های ابرلگ‌لگ در پایگاه‌های داده شناخته شده‌ای مانند Amazon Redshift، Redis، Apache CouchDB، Riak و جز اینها پیاده‌سازی شده‌اند.

الگوریتم ابرلگ‌لگ++

اشتفان هویله، مارک نونکسر، و الکساندر هال در سال ۱۳۹۲، نسخه بهبود یافته ابرلگ‌لگ را به نام ابرلگ‌لگ++ معرفی کردند که بر روی تعداد منحصربه‌فردهای بزرگ و تصحیح سوگیری بهتر متمرکز است. اهم اصلاحات در الگوریتم ابرلگ‌لگ++ استفاده از تابع درهم‌ساز ۶۴ بیتی است. واضح است که هرچه مقادیر خروجی تابع درهم‌ساز طولانی‌تر باشد، مقادیر مختلف بیشتری را می‌توان کدگذاری کرد. چنین بهبودی امکان تخمین تعداد منحصربه‌فردهای بسیار بزرگتر از $۱۰^۹$ مقدار منحصربه‌فرد را فراهم می‌کند، اما زمانی که تعداد منحصربه‌فرد به $۱.۸۰ \cdot ۱۰^{۱۹} \approx ۲^{۶۴}$ نزدیک می‌شود، تصادم‌های درهم برای ابرلگ‌لگ++ نیز مشکل‌ساز می‌شوند.

الگوریتم ابرلگ‌لگ++ عینا از تابع ارزیابی (۹) استفاده می‌کند. با این حال، تصحیح سوگیری را بهبود می‌بخشد. نویسندگان الگوریتم مجموعه‌ای از آزمایش‌ها را برای اندازه‌گیری سوگیری انجام دادند و دریافتند که برای $n \leq ۵m$ سوگیری الگوریتم ابرلگ‌لگ صلی را می‌توان با استفاده از داده‌های تجربی جمع‌آوری شده در طول آزمایش‌ها، تصحیح بیشتر کرد.

علاوه بر مقاله اصلی، هویله و همکاران مقادیر تعیین شده تجربی را در آرایه‌هایی از تخمین‌های خام تعداد منحصربه‌فرد داده تخمینی خام RAWESTIMATEDATA و سوگیری‌های مرتبط را در داده‌سوگیری BIASDATA برای بهبود تصحیح سوگیری در الگوریتم عرضه کردند. البته، پوشش هر مورد بالقوه ممکن نیست،در نتیجه RAWESTIMATEDATA آرایه‌ای از ۲۰۰ نقطه درون‌یابی را در بر دارد که میانگین تخمین خام اندازه‌گیری شده در این نقطه را در بیش از ۵۰۰۰ مجموعه داده مختلف ذخیره دارد. BIASDATA حدود ۲۰۰ سوگیری اندازه‌گیری شده را شامل می‌شود که با RAWESTIMATEDATA مطابقت دارند. هر دو آرایه صفر-اندیس هستند و حاوی مقادیر از پیش محاسبه شده برای همه دقت‌های پشتیبانی شده $p \in ۴ \ldots ۱۸$ هستند، جایی که اندیس صفر در آرایه‌ها با مقدار دقت ۴ مطابقت دارد. به عنوان مثال، برای $m = ۲^{۱۰}$ و p = ۱۰ داده‌های مورد نیاز را می‌توان در $RAWESTIMATEDATA[۶]$ و $BIASDATA[۶]$ یافت. روش تصحیح سوگیری در الگوریتم ابرلگ‌لگ++ را می‌توان به صورت زیر انجام داد.

الگوریتم ۱۰- تصحیح سوگیری با استفاده از مقادیر تجربی
ورودی: تخمین $\hat{n}$ با دقت p
خروجی: تخمین تعداد منحصربه‌فرد تصحیح شده با سوگیری
$n_{low} \leftarrow ۰, n_{up} \leftarrow ۰, j_{low} \leftarrow ۰, j\_up \leftarrow ۰$
برای $j \leftarrow ۰$ تا طول(RAWESTIMATEDATA [p - ٤]) انجام بده
  اگر $\hat{n} \geq RAWESTIMATEDATA\,[p - ۴][j]$ آنگاه
    $j_{low} \leftarrow j - ۱, j_{up} \leftarrow j$
    $n_{low} \leftarrow RAWESTIMATEDATA[p - ۴][j_{low}]$
    $n_{up} \leftarrow RAWESTIMATEDATA[p - ۴][j_{up}]$
    $break$
$b_{low} \leftarrow BIASDATA[p - ۴][j_{low}]$

$b_{up} \leftarrow BIASDATA[p - ۴][j_{up}]$
$y = \text{درون‌یابی} \left((n_{low}, n_{low} - b_{low}), (n_{up}, n_{up} - b_{up})\right)$
برگرداندن $y(\hat{n})$

مثال ۱۰- تصحیح سوگیری با استفاده از مقادیر تجربی
فرض کنید که تخمین تعداد منحصربه‌فرد $\hat{n} = ۲۰۱۸.۳۴$ را با استفاده از معادله (۳.۱۲) محاسبه کرده‌ایم و می‌خواهیم آن را برای دقت $p = ۱۰$ $(m = ۲^{۱۰})$ تصحیح کنیم.
ابتدا، آرایه $RAWESTIMATEDATA[۶]$ را بررسی می‌کنیم و تعیین می‌کنیم که مقدار مذکور $\hat{n}$ بین مقادیر با اندیس‌های ۷۳ و ۷۴ آن آرایه قرار می‌گیرد، جایی که $RAWESTIMATEDATA[۶][۷۳] = ۲۰۰۳.۱۸۰۴$ و $RAWESTIMATEDATA[۶][۷۴] = ۲۰۲۶.۰۷۱$ است.

$$۲۰۰۳.۱۸۰٤ \leq \hat{n} \leq ۲۰۲۶.۰۷۱$$

بنابراین، باید سوگیری‌ها را از $BIASDATA[۶]$ که در همان موقعیت‌های ۷۳ و ۷۴ اندیس شده‌اند، بازیابی کنیم که عبارتند از $BIASDATA[۶][۷۳] = ۱۳۴.۱۸۰۴$ و $BIASDATA[۶][۷۴] = ۱۳۱.۰۷۱$.
تخمین صحیح در بازه زیر است:

$$[۲۰۰۳.۱۸۰۴ - ۱۳۴.۱۸۰۴, ۲۰۲۶.۰۷۱ - ۱۳۱.۰۷۱] = [۱۸۶۹.۰, ۱۸۹۵.۰]$$

و برای محاسبه تقریب تصحیح شده، می‌توانیم آن مقادیر را درون‌یابی کنیم، به عنوان مثال با استفاده از جستجوی k-نزدیک‌ترین همسایه یا فقط با یک درون‌یابی خطی $y(x) = ax + b$، که در آن $y(۲۰۰۳,۱۸۰٤) = ۱۸٦۹,۰$ و $y(۲۰۲٦,۰۷۱) = ۱۸۹٥,۰$.
بنابراین، با استفاده از محاسبات ساده، خط درون‌یابی است

$$y = ۱.۱۳۵۸۳۷\, x - ۴۰۶.۲۸۷۲۵$$

و مقدار درون‌یابی شده برای تخمین تعداد منحصربه‌فرد ما است

$$n = y(\hat{n}) = y(۲۰۱۸.۳۴) \approx ۱۸۸۶.۲۱۸$$

با توجه به آزمایش‌های نویسندگان $HyperLogLog++$، تخمین n_{lin} که بر اساس الگوریتم شمارش خطی ساخته شده است، حتی در مقایسه با مقدار تصحیح شده با سوگیری n برای تعداد منحصربه‌فردهای کوچک هنوز بهتر است. بنابراین، اگر حداقل یک شمارنده خالی وجود داشته باشد، الگوریتم علاوه بر این تخمین خطی را محاسبه می‌کند و از فهرستی از آستانه‌های تجربی، که در جدول ۷ می‌توان یافت، برای انتخاب ارزیابی ترجیحی استفاده می‌کند. در چنین حالتی، مقدار تصحیح شده با سوگیری n تنها زمانی استفاده می‌شود که تخمین خطی n_{lin} بالاتر از آستانه χ_m برای m فعلی قرار گیرد.

مثال ۱۱- تصحیح سوگیری با آستانه
مثال ۱۰ را در نظر می‌گیریم، که در آن برای $m = ۲^{۱۰}$ مقدار تصحیح شده با سوگیری $n \approx ۱۸۸۶.۲۱۸$ را محاسبه کردیم. برای تعیین اینکه آیا باید این مقدار را به تخمین شمارش خطی ترجیح دهیم یا خیر، باید تعداد شمارنده‌های خالی Z را در داده‌ساختار ابرلگ‌لگ++ پیدا کنیم. از آنجایی که آن مقدار را در مثال خود نداریم، فرض می‌کنیم مقدار مذکور برابر $Z = ۷۳$ است. بنابراین، تخمین خطی طبق معادله تخمین برابر است با:

$$n_{lin} = ۲^{۱۰} \cdot \log\left(\frac{۲^{۱۰}}{۷۳}\right) \approx ۲۷۰۴$$

سپس، n_{lin} را با آستانه $\chi_m = ۹۰۰$ از جدول ۷ مقایسه می‌کنیم که بسیار کمتر از مقدار محاسبه شده است، بنابراین تخمین تصحیح شده با سوگیری n را به تخمین شمارش خطی n_{lin} ترجیح می‌دهیم.

جدول ۷- آستانه‌های تجربی χ_m برای مقادیر دقت پشتیبانی شده

p	m	χ_m	p	m	χ_m	p	m	χ_m
۴	$۲^۴$	۱۰	۹	$۲^۹$	۴۰۰	۱۴	$۲^{۱۴}$	۱۱۵۰۰
۵	$۲^۵$	۲۰	۱۰	$۲^{۱۰}$	۹۰۰	۱۵	$۲^{۱۵}$	۲۰۰۰۰
۶	$۲^۶$	۴۰	۱۱	$۲^{۱۱}$	۱۸۰۰	۱۶	$۲^{۱۶}$	۵۰۰۰۰
۷	$۲^۷$	۸۰	۱۲	$۲^{۱۲}$	۳۱۰۰	۱۷	$۲^{۱۷}$	۱۲۰۰۰۰
۸	$۲^۸$	۲۲۰	۱۳	$۲^{۱۳}$	۶۵۰۰	۱۸	$۲^{۱۸}$	۳۵۰۰۰۰

الگوریتم کامل ابرلگ‌لگ++ در زیر نشان داده شده است.

الگوریتم ۱۱- تخمین تعداد منحصربه‌فرد با ابرلگ‌لگ++

ورودی: مجموعه داده D

ورودی: آرایه‌ای از m شمارنده لگ‌لگ با تابع درهم‌ساز h

خروجی: تخمین تعداد منحصربه‌فرد

$COUNTER\,[j] \leftarrow ۰, j = ۰ \dots m - ۱$

برای $x \in D$ انجام بده

$i \leftarrow h(x) = (i_{۶۳} \dots i_۱ i_۰)_۲$

$j \leftarrow (i_{p-۱} \dots i_۱ i_۰)_۲$

$r \leftarrow rank\left((i_{۶۳} \dots i_{p+۱} i_p)_۲\right)$

$COUNTER[j] \leftarrow max(COUNTER[j], r)$

$R \leftarrow \sum_{k=۰}^{m-۱} ۲^{-Counter[j]}$

$\hat{n} = \alpha_m \cdot m^۲ \cdot \frac{۱}{R}$

$n \leftarrow \hat{n}$

اگر $\hat{n} \leq ۵m$ آنگاه

$n \leftarrow CorrectBias\,(\hat{n})$

$Z \leftarrow$ تعداد$_{j=۰\dots m-۱}\,(COUNTER\}[j] = ۰)$

اگر $Z \neq ۰$ آنگاه

$n_{lin} \leftarrow m \cdot \log\frac{m}{z}$

اگر $n_{lin} \leq \chi_m$ آنگاه

$n \leftarrow n_{lin}$

برگرداندن n

ویژگی‌ها

دقت ابرلگ‌لگ++ در طیف وسیعی از محاسبات تعداد منحصربه‌فردها بهتر از ابرلگ‌لگ است و برای بقیه به همان اندازه خوب است. برای تعداد منحصربه‌فردهای بین ۱۲۰۰۰ و ۶۱۰۰۰، تصحیح سوگیری امکان خطای کمتری را فراهم می‌کند و از افزایش ناگهانی خطا هنگام جابجایی بین الگوریتم‌های فرعی جلوگیری می‌کند.

با این حال، به دلیل عدم نیاز ابرلگ‌لگ++ به ذخیرهٔ مقادیر درهم‌ساز، فقط یک به اضافه حداکثر اندازه تعداد صفرهای ابتدایی، حافظه موردنیاز در مقایسه با ابرلگ‌لگ رشد قابل‌توجهی نمی‌کند و با توجه به معادله تخمین حافظه به ۶×۲^p بیت نیاز دارد. الگوریتم

ابرلگ‌لگ++ تخمین تعداد منحصربه‌فرد حدود ۷.۹×۱۰^{۹} مقدار با میزان خطای معمولی ۱.۶۲۵درصد با استفاده از ۲.۵۶ کیلوبایت حافظه استفاده می‌شود.

همانطور که قبلا اشاره شد، الگوریتم از رویکرد میانگین‌گیری تصادفی استفاده می‌کند و مجموعه داده را به $m = ۲^{p}$ زیرمجموعه $\{S_j\}_{j=۰}^{m-۱}$ تقسیم می‌کند که هر کدام دارای شمارنده‌های مرتبط $\{COUNTER[j]\}_{j=۰}^{m-۱}$ هستند، هر شمارنده اطلاعات مربوط به $\frac{n}{m}$ مقدار را مدیریت می‌کند. هویله و همکاران متوجه شدند که برای $n \ll m$ بیشتر شمارنده‌ها هرگز استفاده نمی‌شوند و نیازی به ذخیره‌سازی ندارند، بنابراین ذخیره‌سازی می‌تواند از نمایشی تنک بهره گیرد. اگر تعداد منحصربه‌فرد n بسیار کوچکتر از m باشد، ابرلگ‌لگ++ به حافظه قابل توجهی کمتری نسبت به روش‌های قبلی خود نیاز دارد.

الگوریتم ابرلگ‌لگ++ در نسخه تنک صرفا جفت‌های $(j, COUNTER[j])$ را ذخیره می‌کند و آنها را با الحاق الگوهای بیتی خود به عنوان یک عدد صحیح منفرد نشان می‌دهد. همه این جفت‌ها در فهرست مرتب شده منفرد از اعداد صحیح ذخیره می‌شوند. چون همیشه در پی محاسبهٔ حداکثر رتبه هستیم، نیازی به ذخیره جفت‌های مختلف با اندیس یکسان نیست. در عوض فقط جفتی با حداکثر اندیس باید ذخیره شود. در عمل، برای ارائه نتیجهٔ بهتر، می‌توان فهرست مرتب نشده دیگر را برای درج‌های سریع حفظ کرد که باید به طور دوره‌ای مرتب شده و در فهرست اصلی ادغام شوند. اگر چنین فهرستی به حافظه بیشتری نسبت به نمایش متراکم شمارنده‌ها نیاز داشته باشد، می‌توان به راحتی به شکل متراکم تبدیل شد. علاوه بر این، برای نمایش تنک فشرده‌تر، الگوریتم ابرلگ‌لگ++ تکنیک‌های فشرده‌سازی مختلفی را با استفاده از رمزگذاری با طول متغیر و رمزگذاری تفاوت برای اعداد صحیح پیشنهاد می‌کند، بنابراین فقط اولین جفت و تفاوت‌ها از مقدار آن ذخیره می‌شوند. در حال حاضر، الگوریتم ابرلگ‌لگ++ به طور گسترده در بسیاری از برنامه‌های پرکاربرد، از جمله Google BigQuery و Elasticsearch استفاده می‌شود.