بسامد و تکرار – از اکثریت تا پربسامد

محسن هوشمند

فهرست

بسامد و تکرار

بسیاری از مسائل مهم در برنامه‌های جریانی که با جریان‌های داده بزرگ کار می‌کنند، به تخمین بسامد مقادیر از جمله تعیین پربسامدترین مقدار یا تشخیص مقادیر روند در بازه زمانی مربوط می‌شوند. همانطور که در مسائل دیگر دیدیم، زمانی که جریان‌های داده به اندازه کافی بزرگ هستند (می‌توان آنها را به عنوان دنباله‌ای بی‌نهایت از مقادیر در نظر گرفت) و تعداد مقادیر متمایز زیادی دارند، راه‌حل‌های معمول، مانند مرتب‌سازی یا نگه داشتن شمارنده برای هر عنصر، دیگر امکان‌پذیر نیستند. همچنین، توجه به این نکته مهم است که در بیشتر موارد ذخیره و پردازش مجدد چنین دنباله‌هایی امکان‌پذیر نیست، بنابراین الگوریتم‌های جریان داده تک‌گذره مورد نیاز هستند.

اگر جریان داده بزرگ باشد اما تعداد منحصربه‌فرد پایینی داشته باشد (فقط شامل تعداد کمی از مقادیر متمایز باشد)، حفظ شمارنده‌های دقیق بسامد با استفاده از یک شمارنده به ازای هر مقدار متمایز کافی است، در این موارد نیازی به الگوریتم‌های خاص نیست.

ویژگی خاص برنامه‌های داده بزرگ که جریان‌های داده بزرگ را مدیریت می‌کنند، ایجاب می‌کند که ساختارهای داده و الگوریتم‌های مناسب شرایطی را برآورده کنند که عبارتند از: الف- عملیات را در یک گذر از داده‌ها انجام دهند. ب- فضای زیرخطی (حداکثر چندلگاریتمی) داشته باشند، به این معنی که به اندازه جریان ورودی به سرعت رشد نمی‌کنند. ج-از به‌روزرسانی‌های سریع و ساده با تضمینی از دقت پشتیبانی کنند.

به دلیل محدودیت‌های فضا، واضح است که چنین ساختارهایی باید داده‌ها را به صورت فشرده عمل کنند که خلاصه‌ای از جریان داده است (به عنوان مثال، طرحواره) و محاسبه دقیق بیشتر توابع روی جریان را غیرممکن می‌سازد، بنابراین تقریب احتمالاتی مورد نیاز است.

منظور از جریان داده $D = \{x_1, x_2, \dots, x_n\}$ دنباله‌ای از مقادیر با هر ماهیتی است، با این فرض که تعداد مقادیر n بسیار زیاد است، مثلا میلیاردها، و تعداد زیادی ناشناخته از مقادیر متمایز وجود دارد. اگر جریان واقعا بی‌نهایت باشد، D را می‌توان در صورت مشاهده در یک پنجره زمانی به عنوان زیرجریان در نظر گرفت. با رویکردی برای تخمین بسامد مقادیر در جریان داده عظیم، می‌توان مسئله رایج یافتن فهرست مقادیر با بسامد بالا در جریان را که به عنوان مسئله پربسامد شناخته می‌شود، بررسی کرد.

وقتی به دنبال مقداری هستیم که بیش از $\frac{n}{2}$ دفعه در جریان داده D رخ می‌دهد، مسئله اکثریت را در نظر می‌گیریم که جی. استروتر مور در مجله الگوریتم‌ها در سال ۱۳۶۰ به عنوان مسئله‌ای تحقیقاتی تدیون و معرفی کرد. می‌توانیم فرض کنیم که چنین مقداری در جریان وجود دارد، که همیشه صادق نیست، و با تعریف، واضح است که فقط می‌تواند یک مقدار برای جریان داده وجود داشته باشد که مقدار اکثریت نامیده می‌شود.

مسئله مذکور راه‌حلی نهائی نیست، بلکه درک بهتری از مسائل بسامد مربوط به جریان‌های داده به دست می‌دهد.

از پیچیده‌ترین مسائلی که در عمل هنگام کار با داده بزرگ ظاهر می‌شود، مسئله یافتن k مقدار پربسامد در جریان است که بیش از $\frac{n}{k}$ بار رخ می‌دهند، همچنین به عنوان مقادیر پربسامد شناخته می‌شوند، که در آن $k \ll n$ و معمولا برابر ۱۰، ۱۰۰ یا ۱۰۰۰ هستند. جستجوی مقادیر پربسامد فرض می‌کند که برخی از مقادیر می‌توانند به طور قابل توجهی بیشتر از سایرین در جریان داده رخ دهند، در غیر این صورت حل این مسئله معنا ندارد.

با روش‌هائی از پیچیدگی ارتباطی ثابت شده است که هیچ الگوریتمی که بتواند مسئله مقادیر پربسامد را در یک گذر با استفاده از فضای زیرخطی حل کند وجود ندارد.

بسیاری از کاربردهای عملی با مسئله مقادیر پربسامد مرتبط هستند، از جمله جستجو، استخراج لاگ، تحلیل شبکه، مهندسی ترافیک و تشخیص ناهنجاری. برای مثال، ممکن است بخواهیم k کاربر پرکارتر (برای مقدار دلخواه $k \ll n$) را برای تارمانهٔ با ترافیک بالا تعیین کنیم. با این حال، ممکن است برخی از کاربران بار تقریبا مساوی داشته باشند و دریافت پاسخ دقیق به این سوال با استفاده از فضای محدود غیرممکن است.

در عمل، مسئله ϵ-مقادیر پربسامد را به عنوان تقریب-ϵ از مسئله مقادیر پربسامد در نظر می‌گیریم که منجر به مقادیری می‌شود که حداقل $\frac{n}{k}$ بار رخ می‌دهند این تعداد تکرار با وقوع تضمین شده حداقل $\frac{n}{k} - \epsilon \cdot n$، که در آن $\epsilon > ۰$ و کوچک است. برای مثال، برای $\epsilon = \frac{۱}{۲k} > ۰$ خروجی مسئله ϵ -مقادیر پربسامد عناصری با بسامد حداقل $\frac{n}{k}$ و تضمینی که حداقل $\frac{n}{k} - \epsilon \cdot n = \frac{n}{k} - \frac{n}{۲k} = \frac{n}{۲k}$ بار رخ می‌دهند، خواهد بود.

برای جریان‌های داده کوچک (بدون توجه به تعداد مقادیر منحصربه‌فرد) کافی است فقط مقادیر را مرتب کنیم و با پویشی خطی، مقادیری را پیدا کنیم که حداقل $\frac{n}{k}$ بار رخ می‌دهند. اینها مقادیر پربسامد خواهند بود.

مسئله اکثریت را می‌توان به عنوان حالتی خاص از مسئلهٔ مقادیر پربسامد با این شرط که چنین مقدار اکثریتی وجود داشته باشد و $k \approx ۲ - \epsilon$، که در آن $\epsilon > ۰$ و کوچک است، در نظر گرفت.

مثال ۱: تشخیص حمله DDoS DNS: حمله انکار سرویس توزیع شده (DDoS) شامل بسیاری از سیستم‌ها است که منابع سیستم هدف را معمولا با ارسال تعداد زیادی پرس و جو از باتنتی سیل‌آسا اشغال می‌کنند. یکی از اهداف، سیستم نام دامنه (DNS) است که نقش «دفترچه تلفن» اینترنت را بازی می‌کند و ترجمه بین نام‌های دامنه به یاد ماندنی و آدرس‌های IP وب‌سایت‌ها را فراهم می‌کند.
پرس و جوهای DNS به عنوان جریان داده در نظر گرفته می‌شوند که در آن هر مقدار دارای دامنه‌ای مرتبط برای حل است. با ادامه کار می‌توانیم پرس و جوها را با استفاده از دامنه سطح بالای آنها گروه‌بندی کنیم و با بررسی سنگین‌ترین دامنه‌ها در جریان پرس و جو می‌توانیم سیل DNS تصادفی را زمانی که پرس و جوها برای بسیاری از زیردامنه‌های مختلف غیر موجود از همان دامنه اصلی صادر می‌شوند، تشخیص دهیم.
مسئله «حداکثر تغییر»، از دیگر عملیات‌های محل توجه در برنامه‌های جریانی است که عناصری را تعیین می‌کند که بسامد آنها در جریان‌های داده مختلف یا پنجره‌های زمانی بیشترین تغییر را داشته است. این مسئله برای موتورهای جستجو اهمیت عملی دارد زیرا پرس و جوهایی که بسامد آنها بین دو دوره زمانی متوالی بیشترین تغییر را دارد می‌تواند نشان دهد که کدام موضوعات با سریعترین سرعت در حال افزایش یا کاهش محبوبیت هستند.

مثال ۲- هشتگ‌های ترند توییتر: از هشتگ برای نمایه‌سازی موضوع در توییتر و دیگر رسانه‌ها استفاده می‌شود و به افراد امکان می‌دهد به راحتی موارد مورد علاقه خود را دنبال کنند. هشتگ‌ها معمولا با نماد # در ابتدا نوشته می‌شوند.
مطابق گزارشات هر ثانیه حدود ۶۰۰۰ توییت در توییتر ایجاد می‌شود، که معادل تولید تقریبا ۵۰۰ میلیارد پیام در روز است. بیشتر این توییت‌ها با یک یا چند هشتگ مرتبط هستند و برای آگاهی از همه رویدادهای اخیر، تعیین محبوب‌ترین موضوعات روز مهم است.
این کار را می‌توان با پردازش جریان داده توییت‌ها، تخمین بسامد هر هشتگ و یافتن پربسامدترین مقادیر انجام داد. علاوه بر این، ممکن است مفید باشد که بسامدهای دیروز و امروز را برای تعیین موضوعاتی که در حال روند هستند، مقایسه کنیم، به عنوان مثال، موضوعاتی که بیشترین افزایش بسامد را از دیروز داشته‌اند.

در ادامه، رویکردهای مختلفِ حل مسائل مربوط به بسامد در جریان‌های داده بزرگ معرفی می‌شود. با الگوریتم‌های قطعی بسیار ساده شروع می‌کنیم و پس از آن روش‌های احتمالاتی متاخر را شرح می‌دهیم که مسائل دنیای واقعی را با کارائی بالا حل می‌کنند.

الگوریتم اکثریت

می‌توان راه‌حلی زمان خطی برای مسئله اکثریت پیشنهاد کرد، زیرا مقدار اکثریت (البته با فرض وجود آن) میانه است. نقطه ضعف یافتن میانه در این است که نیاز به چندین گذر از جریان دارد و بنابراین برای جریان‌های داده بزرگ مناسب نیست.

الگوریتم اکثریت، همچنین به عنوان الگوریتم رأی اکثریت بویر-مور شناخته می‌شود، را باب بویر و جی. استروتر مور در سال ۱۳۶۰ برای حل مسئله اکثریت در یک گذر از جریان داده اختراع کردند. راه‌حلی مشابه نیز به طور مستقل توسط مایکل جی. فیشر و استیون ال. سالزبرگ در سال ۱۳۶۱ پیشنهاد شد.

داده‌ساختار الگوریتم اکثریت نسبتا ساده است و یک جفت متشکل از شمارنده صحیح و به اصطلاح مقدار تحت نظر است: $S = (c, x^*)$. بنابراین، به مقدار ثابتی از حافظه نیاز دارد، اما اندازه آن بسته به اندازه مقادیر متفاوت است.

چنین داده‌ساختاری از عملیات به‌روزرسانی پشتیبانی می‌کند، که شمارنده را بر اساس وضعیت قبلی خود و مقدار فعلی x به‌روزرسانی کرده و نامزد مقدار تحت نظر را انتخاب می‌کند.

الگوریتم ۱- به‌روزرسانی داده‌ساختار اکثریت
ورودی: مقدار $x \in D$
اگر $c = ۰$ آنگاه
$x^* \leftarrow x$
اگر $x = x^*$ آنگاه
$c \leftarrow c + ۱$
در غیر این صورت
$c \leftarrow c - ۱$

با این نوع داده‌ساختار، توصیف الگوریتم ساده است. برای هر مقدار x در جریان D روش به‌روزرسانی داده شده در الگوریتم ۱ را فعال می‌کند و با این شرط که مقدار اکثریت وجود داشته باشد، آخرین مقدار تحت نظر را به عنوان مقدار اکثریت بازمی‌گرداند. شایان توجه است که مقدار شمارنده بسامد مقدار اکثریت نیست.

الگوریتم ۲- الگوریتم اکثریت
ورودی: جریان داده D
خروجی: مقدار اکثریت
$c \leftarrow ۰$
$x^* \leftarrow \phi$
برای $x \in D$ انجام بده
$Update(x)$
برگرداندن x^*

در الگوریتم اکثریت، هر مقدار «غیراکثریت» که به دنبال می‌آید می‌تواند شمارنده c را کاهش دهد یا حتی آن را به صفر بازنشانی کند، که موجب می‌شود مقدار تحت نظر x^* دوباره انتخاب شود. با نگاهی سطحی ممکن است مشخص نباشد که چگونه چنین الگوریتمی به مقدار صحیح ختم می‌شود و یا این پرسش مطرح شود که آیا خطر حذف مقادیر اکثریت در بعی از مجموعه داده‌های ورودی وجود ندارد. مقادیر «غیر اکثریت» می‌توانند فقط یک نسخه از مقدار اکثریت قبلی را حذف کنند، اما از آنجایی که باید بیش از $\frac{n}{۲}$ مقدار اکثریت در جریان داده وجود داشته باشد، مقادیر «غیر اکثریت» کافی برای حذف همه وجود نخواهد داشت و حداقل نسخه‌ای از مقدار اکثریت در پایان باقی می‌ماند. چنین مشاهده‌ای روشن می‌کند مقدار برگشتی شمارنده را نمی‌توان به عنوان تقریبی از بسامد مقدار اکثریت استفاده کرد.

هنگامی که مقدار اکثریت وجود نداشته باشد، خروجی الگوریتم اکثریت مقداری دلخواه از جریان داده است. بنابراین، اعمال چنین الگوریتمی زمانی که از وجود مقدار اکثریت مطمئن نیستیم، نیاز به گذر دیگری از جریان داده با شمارنده‌ای معمول دارد تا اکثریت بودن مقدار برگردانده شده با الگوریتم ۲ با تکراری بیش از $\frac{n}{۲}$ بار تائید شود.

مثال۳- الگوریتم اکثریت: مجموعهٔ داده با $n = ۱۰$ مقدار $\{۴,۴,۳,۵,۶,۴,۴,۴,۴,۲\}$ را در نظر می‌گیریم که در آن مقدار اکثریت $x = ۴$ است و شش بار از ده بار رخ می‌دهد.
طبق الگوریتم، جفت $S = (c, x^*) = (۰, \phi)$ را مقداردهی اولیه می‌کنیم و مقادیر مجموعه داده را می‌خوانیم. اولین مقدار $x_۱ = ۴$ است. به دلیل خالی بودن شمارنده c، آن را به عنوان مقدار تحت نظر $x^* = ۴$ ذخیره و شمارنده افزایش می‌یابد و $c = ۱$. مقدار بعدی $x_۲$ دوباره ۴ است که با مقدار تحت نظر برابر است، بنابراین فقط شمارنده را افزایش می‌دهیم: $c = ۲$. سومین مقدار ورودی $x_۳ = ۳$ است که با $x^* = ۴$ متفاوت است، بنابراین شمارنده خود را کاهش می‌دهیم: $c = ۱$. به طور مشابه، پس از پردازش $x_۴ = ۵$، دوباره شمارنده را کاهش می‌دهیم و صفر می‌شود: $c = ۰$.
سپس، مقدار $x_۵ = ۶$ را پردازش می‌کنیم و از آنجایی که مقدار شمارنده فعلی صفر است، مقدار تحت نظر را به $x^* = ۶$ به‌روزرسانی می‌کنیم و شمارنده آن را تنظیم می‌کنیم: $c = ۱$. با این حال، این وضعیت طولانی نمی‌ماند و پس از مدیریت مقادیر $x_۶ = ۴$ و $x_۷ = ۴$، مقدار تحت نظر دوباره $x^* = ۴$ با شمارنده $c = ۱$ می‌شود. مقدار بعدی شمارنده را به $c = ۲$ افزایش می‌دهد، اما آخرین مقدار برابر با ۴ نیست و شمارنده دوباره به مقدار $c = ۱$ کاهش می‌یابد.
در پایان، مقدار اکثریت صحیح ۴ به عنوان مقدار تحت نظر باقی می‌ماند. با این حال، توجه داشته باشید که شمارنده باقی‌مانده c تخمین‌گر بسامد نیست و حاوی مقدار کاملا متفاوتی است.

الگوریتم پربسامد

اریک دی. دمین، الخاندرو لوپز-اورتیز و جی. یان مونرو در سال ۱۳۸۱ الگوریتم پربسامد را که تعمیمی از الگوریتم اکثریت است معرفی کردند. بعدها روشن شد که این الگوریتم در واقع همان الگوریتمی است که جایادیو میسرا و دیوید گریس در سال ۱۳۶۱ معرفی کردند، که اکنون به عنوان الگوریتم میسرا-گریس شناخته می‌شود.

الگوریتم پربسامد برای رسیدگی به مسئله مقادیر پربسامد طراحی شده است و به جای نگه داشتن یک شمارنده مانند الگوریتم اکثریت، داده‌ساختار بسامد از مجموعه‌ای از مقادیر تحت نظر X^* و آرایه‌ای از p شمارنده $C = \{c_i\}_{i=1}^{p}$ تشکیل شده است.

در پردازش مقداری جدید از جریان داده، ابتدا وجود قبلی آن را بررسی می‌شود. اگر مقدار ورودی جدید باشد، به X^* اضافه می‌شود. البته این کار صرفا زمانی انجام می‌پذیرد که فضا پر نشده باشد زیرا حداقل p مقدار را حفظ می‌کنیم. اگر مقدار اضافه نشد، همچنان می‌خواهیم حضور آن را در جریان با کاهش شمارنده‌های همه مقادیر در مجموعه مقادیر تحت نظر منعکس کنیم. وقتی مقدار از قبل در X^* وجود دارد، فقط شمارنده مرتبط با آن را افزایش می‌دهیم. در پایان روش، از فهرست عبور می‌کنیم و همه عناصری را که شمارنده‌هایشان به صفر رسیده است حذف می‌کنیم.

در مقاله اصلی، با اینکه میسرا و گریس از درخت‌های جستجوی متوازن برای نمایش داده‌ساختار استفاده کردند، محققان بعدی از جدول‌های درهم‌ساز استفاده و آن را در قالب لغت‌نامه پیاده‌سازی کردند.

الگوریتم۳- به‌روزرسانی داده‌ساختار پربسامد
ورودی: مقدار $x \in D$
ورودی: داده‌ساختار پربسامد با p شمارنده
اگر $x \notin X^*$ آنگاه
  اگر $\exists\, m: c_m = 0$ آنگاه
    $x_m^* \leftarrow x$
اگر $x \in X^*$ آنگاه
  $\exists\, m: x_m^* = x$
  $c_m \leftarrow c_m + 1$
در غیر این صورت
  برای $j \leftarrow 1$ تا p انجام بده
    اگر $c_j > 0$ آنگاه
      $c_j \leftarrow c_j - 1$
  برای $j \leftarrow 1$ تا p انجام بده
    اگر $c_j = 0$ آنگاه
      $X^* \leftarrow X^* \setminus \{x_j^*\}$

الگوریتم پربسامد از داده‌ساختار به طول p برای کشف مقادیری استفاده می‌کند که حداقل $\frac{n}{p+1}$ بار در جریان داده به طول n رخ می‌دهند. بنابراین، برای تعیین حداکثر $k - 1$ مقدار پربسامد که حداقل $\frac{n}{k}$ بار در جریان داده رخ می‌دهند، باید از $p = k - 1$ شمارنده استفاده کنیم.

الگوریتم ۴- الگوریتم پربسامد

ورودی: جریان داده D
ورودی: داده‌ساختار پربسامد با $k - ۱$ شمارنده
خروجی: مقادیر پربسامد
$C = \{c_i\}_{i=۱}^{k-۱}, c_i \leftarrow ۰$
$X^* \leftarrow \emptyset$
برای $x \in D$ انجام بده
Update(x)
برگرداندن X^*

شهود الگوریتم پربسامد بسیار شبیه به الگوریتم اکثریت با توجه به شرطی است که مقادیر پربسامد بیش از $\frac{n}{k}$ بار رخ می‌دهند.

مثال ۴- یافتن مقادیر پربسامد با الگوریتم پربسامد
جریان داده با n = ۱۸ مقدار {۴,۴,۴,۴,۶,۲,۳,۵,۴,۴,۳,۳,۴,۲,۳,۳,۳,۲} را داریم. یافتن مقادیر پربسامدی که حداقل $\frac{n}{۳} = ۶$ بار در جریان داده رخ می‌دهند، نیاز به داده‌ساختار با p = ۲ شمارنده دارد و از الگوریتم ۴ برای شناسایی حداکثر دو مقدار پربسامد احتمالی استفاده می‌کنیم.

	۱	۲
X^*		
شمارنده	۰	۰

از ابتدای جریان داده مقادیر وارد می‌شوند و اولین مقدار برابر ۴ است. چون X^* تهی است، مقدار ۴ را به مجموعه مقادیر تحت نظر اضافه و شمارنده متناظر را افزایش می‌دهیم، $c_۱ = ۱$.

	۱	۲
X^*	۴	
شمارنده	۱	۰

به طور مشابه، سه مقدار بعدی را که برابر با ۴ هستند خوانده و عملیات متناظر را اعمال می‌کنیم. به دلیل حضور از قبل مقدار در X^*، شمارنده متناظرِ $c_۱$ را افزایش می‌دهیم.

	۱	۲
X^*	۴	
شمارنده	۴	۰

مقدار بعدی ۶ است که هنوز تحت نظر نیست. به دلیل پر نبودن مجموعه X^*، مقدار ۶ را در آن درج و شمارنده متناظر را یک واحد افزایش می‌دهیم، $c_۲ = ۱$.

	۱	۲
X^*	۴	۶
شمارنده	۴	۱

ورودی بعدی مقدار ۲ است که در X^* نیست، و در عین حال مجموعه تحت‌نظر نیز پر شده است و نمی‌توان آن را افزود. شمارنده‌های همه مقادیر موجود در X^* را کاهش می‌دهیم. با توجه به الگوریتم ۴ همچنین باید مقادیری را که شمارنده‌هایشان به صفر رسیده از مجموعه تحت نظر حذف کنیم. در مثال، این مقدار ۶ است که از مجموعه حذف می‌شود.

	۱	۲
X^*	۴	
شمارنده	۳	۰

سپس، مقدار ۳ را از جریان داده مصرف می‌کنیم. این مقدار در X^* نیست و چون ظرفیت خالی دارد، مقدار ۳ را اضافه و شمارنده مرتبط افزایش می‌یابد، $c_۲ = ۱$.

	۱	۲
X^*	۴	۳
شمارنده	۳	۱

در ادامه، به روشی مشابه همه مقادیر باقی‌مانده را پردازش و داده‌ساختار نهایی به این صورت می‌شود:

	۱	۲
X^*	۴	۳
شمارنده	۳	۳

بنابراین، مقادیر پربسامد شناسایی شده ۴ و ۳ هستند. با این حال، شمارنده‌ها بسامدهای واقعی مقادیر در جریان داده را منعکس نمی‌کنند، همانطور که برای الگوریتم اکثریت نیز چنین بود.

ویژگی‌ها

هزینه زمانی الگوریتم تحت سلطه عملیات لغت‌نامه‌ای $O(۱)$ در هر به‌روزرسانی و هزینه کاهش شمارنده‌ها است. برای بهینه‌سازی سرعت الگوریتم، می‌توان همه شمارنده‌ها را یکباره و در زمان ثابت با سازماندهی آنها به ترتیب مرتب شده و استفاده از کدگذاری تفاضلی کاهش داد. به بیان دیگر، اطلاعات ذخیره شده در اختلاف شمارنده‌ها را یکدیگر نشان می‌دهد. روش دیگر، تلاش بر کمینه‌سازی حرکات هنگام افزایش و کاهش شمارنده است که به معنای گروه‌بندی همه شمارنده‌های با مقدار یکسان است. با چنین داده‌ساختار بهینه‌شده، هر شمارنده دیگر نیازی به ذخیره مقدار ندارد، بلکه گروه خود را ذخیره می‌کند. بنابراین، الگوریتم پربسامد را می‌توان به گونه‌ای بهبود داد که در زمان $O(۱)$ اجرا شود.

در واقع، حتی بدون هیچ رویکرد احتمالاتی، الگوریتم حداکثر $k - ۱$ نامزد را در مسئله مقادیر پربسامد ولی صرفا اسامی آنها بدون تقریبی از تعداد تکرار متناظر را تحویل می‌دهد. در صورت نیاز به تخمین بسامد مقادیر، گذر دوم از جریان داده مورد نیاز است که در بیشتر موارد مدیریت جریان‌های داده عظیم غیرقابل اجرا است.

در ادامه، راه‌حل‌های مسائل مربوط به بسامد با ساختارهای داده احتمالاتی بسیار مؤثر که کاملا برای جریان‌های داده بزرگ مناسب هستند را بررسی می‌کنیم

طرحوارهٔ شمارش

طرحواره شمارش الگوریتمی فضا-کارآمد را موسی چاریکار، کوین چن و مارتین فاراچ-کلتون در سال ۱۳۸۱ برای حل بسیاری از مسائل جریان داده مربوط به بسامد پیشنهاد دادند. آنها به داده‌ساختاری فضا-کارآمد نیاز عملی داشتند که بتواند به راحتی شمارش‌های تقریبی مقادیر با بسامد بالا را در جریان داده حفظ کند.

برای درک بهتر مسئله‌ای که طرحواره شمارش حل می‌کند، می‌توان نشان داد که از فیلتر بلوم شمارنده نیز در محاسبه بسامد مقادیر در جریان داده استفاده کرد، با این حال، برای ساختن تخمین‌گرهای بسامد دقیق کافی نیست.

یک داده‌ساختار با آرایه $C = \{c^i\}_{i=۱}^{m}$ از m شمارنده و p تابع درهم‌ساز $h_۱, h_۲, \dots, h_p$ که از مقادیر به $\{۱,۲, \dots, m\}$ نگاشت می‌دهند را در نظر می‌گیریم. اندیس کردن مقدار x از جریان داده در چنین داده‌ساختاری، مانند فیلتر بلوم شمارنده، شامل محاسبه $\{h_j(x)\}_{j=۱}^{p}$ و افزایش شمارنده‌های متناظر $c^{h_j(x)}, j = ۱ \dots p$ در آرایه است.

وقتی نیاز به یافتن بسامد f(x) مقدار x داریم، مقادیر هر تابع درهم‌ساز را برای آن مقدار محاسبه کرده و مقادیر شمارنده‌های متناظر $c^۱, c^۲, \dots, c^m$ را به دست می‌آوریم که نقش تخمین‌های بسامد را بازی می‌کنند.
با این حال، به دلیل خاصیت غیرکاهشی شمارنده‌ها و استفاده از آرایه‌های یکسان توابع درهم‌ساز، واضح است که چنین تخمین‌هایی بزرگتر از بسامد واقعی f(x) مقدار خواهند بود:

$$f(x) \leq c^i, i = ۱ \dots m,$$

کران بالا ناشی از تصادم‌های محتمل درهم حین به‌روزرسانی مقادیر شمارنده‌ها است. به عبارت دیگر، خطای یک‌طرفه رایج در تخمین‌ها باعث می‌شود همه آنها تخمین‌های کران بالا باشند.

طرحواره شمارش بر اساس تخمین کران‌های پایین و بالا است. جهت اجتناب از مواقعی که تصادم مقادیر بسامد بالا تخمین مقادیر بسامد پایین را خراب می‌کند، انتخابی تصادفی جهت کاهش یا افزایش شمارنده صورت می‌پذیرد. به منظور کاهش وردائی، میانه آن تخمین‌ها را می‌گیرد.

داده‌ساختاری که برای ذخیره بسامدهای m مقدار با بسامد بالا طراحی شده است، از آرایه‌ای $p \times m$ از شمارنده‌های $\{c_j^i\}$ تشکیل شده است که می‌توان آن را به عنوان آرایه‌ای از p جدول درهم‌ساز، هر کدام دارای m سطل فرض کرد. مضافا، از p تابع درهم‌ساز $h_۱, h_۲, \dots, h_p$ استفاده می‌کند که مقادیر ورودی را به $\{۱,۲, \dots, m\}$ نگاشت و از p تابع درهم‌ساز دیگر $s_۱, s_۲, \dots, s_p$ که مقادیر را به $\{+۱, -۱\}$ نگاشت می‌دهند تا از تقریب دو طرفهٔ مقدار بسامد واقعی پشتیبانی کنند. فرض می‌شود که یک به یک توابع درهم‌ساز h_i و s_i مستقل و مستقل از یکدیگر هستند.

داده‌ساختار به شمارنده‌ها اجازه می‌دهد برای هر مقدار نمایه شده به‌روزرسانی شوند و تعداد دفعاتی که مقدار در گذشته دیده شده است را تخمین می‌زند، که به عنوان تخمین بسامد برای مقدار استفاده می‌شود. هر بار که یک مقدار جدید x را اندیس می‌کنیم، شمارنده‌های $c_j^{h_j(x)}$ برای هر ردیف j از طرح فشرده می‌توانند بر اساس مقادیر $s_j(x)$ یا افزایش یا کاهش یابند. بنابراین، ممکن است شمارنده‌ها بسامد مقدار x را بیش از حد و یا کمتر از حد تخمین بزنند.

الگوریتم ۵- به‌روزرسانی طرحواره شمارش
ورودی: مقدار $x \in D$
ورودی: طرحواره شمارش با شمارنده‌های $p \times m$
برای $j \leftarrow ۱$ تا p انجام بده
$i \leftarrow h_j(x)$
$c_j^i \leftarrow c_j^i + s_j(x)$

با فرض اینکه هر تابع درهم‌ساز $\{h_j\}_{j=۱}^{p}$ و $\{s_j\}_{j=۱}^{p}$ را می‌توان در زمان ثابت محاسبه کرد، زمان اجرای روش به‌روزرسانی الگوریتم ۵ O(p) است.

مثال ۵- ساختن طرحواره شمارش: مجموعه دادهٔ با n = ۱۸ مقدار {۴,۴,۴,۴,۲,۳,۵,۴,۶,۴,۳,۳,۴,۲,۳,۳,۳,۲} را در نظر می‌گیریم. داده‌ساختار از $m = ۵$ شمارنده با استفاده از p = ۳ تابع درهم‌ساز مبتنی بر مورمور۳، فـن‌و۱الف، و MD۵ برای تصمیم‌گیری در مورد اینکه کدام شمارنده را به‌روزرسانی کنیم، بسازیم:

$$h_۱(x) = MurmurHash3(x)\%\ ۵ + ۱,$$
$$h_۲(x) = FNV1a(x)\%\ ۵ + ۱,$$
$$h_۳(x) = MD5(x)\%\ ۵ + ۱,$$

و سه تابع درهم‌ساز برای تعیین جهت به‌روزرسانی:

$$s_۱(x) = MurmurHash3(x)\%\ ۲\ ?\ -۱ : ۱,$$
$$s_۲(x) = FNV1a(x)\%\ ۲\ ?\ -۱ : ۱,$$
$$s_۳(x) = MD5(x)\%\ ۲\ ?\ -۱ : ۱.$$

در ابتدا، داده‌ساختار از صفر تشکیل شده است:

	۱	۲	۳	۴	۵
$h_۱$	۰	۰	۰	۰	۰
$h_۲$	۰	۰	۰	۰	۰
$h_۳$	۰	۰	۰	۰	۰

اولین مقدار مجموعه داده جهت پردازش ۴ است. با توجه به الگوریتم ۵، مقادیر درهم‌ساز آن $h_۱(۴)$، $h_۲(۴)$ و $h_۳(۴)$ را محاسبه می‌کنیم تا شمارنده‌هایی را که باید به‌روزرسانی شوند تعیین کنیم:

$$i_۱ = h_۱(۴) = ۳, i_۲ = h_۲(۴) = ۳, i_۳ = h_۳(۴) = ۱.$$

در این مورد، دو تابع درهم‌ساز مقدار یکسانی را ارائه می‌دهند، اما به دلیل نگه‌داری فهرست‌های اختصاص هر شمارنده به تابع متناظر درهم‌ساز، مشکلی پیش نمی‌آید. برای تعیین جهت به‌روزرسانی، مقادیر درهم‌ساز $s_۱(۴)$، $s_۲(۴)$ و $s_۳(۴)$ را محاسبه می‌کنیم:

$$s_۱(۴) = ۱, s_۲(۴) = ۱, s_۳(۴) = -۱.$$

بنابراین، شمارنده‌های $c_۱^۳$ و $c_۲^۳$ را افزایش و شمارنده $c_۳^۱$ را کاهش می‌دهیم. داده‌ساختار به صورت زیر در می‌آید.

	۱	۲	۳	۴	۵
$h_۱$	۰	۰	۱	۰	۰
$h_۲$	۰	۰	۱	۰	۰
$h_۳$	−۱	۰	۰	۰	۰

سه مقدار بعدی نیز ۴ هستند، بنابراین همان شمارنده‌ها را سه بار دیگر افزایش یا کاهش می‌دهیم:

	۱	۲	۳	۴	۵
$h_۱$	۰	۰	۴	۰	۰
$h_۲$	۰	۰	۴	۰	۰
$h_۳$	−۴	۰	۰	۰	۰

مقدار بعدی در مجموعه داده ۲ است و اندیس‌های متناظر آن $i_۱ = ۳$، $i_۲ = ۲$ و $i_۳ = ۳$ هستند. مقادیر توابع درهم‌ساز جهت $s_۱(۲) = ۱$، $s_۲(۲) = ۱$ و $s_۳(۲) = -۱$ هستند، بنابراین شمارنده‌های $c_۱^۳$ و $c_۲^۲$ را افزایش و $c_۳^۳$ را کاهش می‌دهیم. در خور

ذکر است تصادمی نرم رخ می‌دهد و مقدار ۲ شمارنده‌ای را که مقدار ۴ استفاده می‌کند (در همان جهت) تغییر می‌دهد. چنین تصادمی موجب می‌شود مقدار موجود در شمارنده $c_{۱}^{۳}$ تخمین تکرار واقعی هر دو مقدار را بیش از حد تخمین بزنند.

	۱	۲	۳	۴	۵
$h_۱$	۰	۰	۵	۰	۰
$h_۲$	۰	۱	۴	۰	۰
$h_۳$	−۴	۰	−۱	۰	۰

به همین ترتیب، همه مقادیر باقی‌مانده را پردازش می‌کنیم. برای مقدار ۳ شمارنده‌های $c_{۱}^{۱}$ و $c_{۲}^{۳}$ را کاهش و شمارنده $c_{۳}^{۴}$ را افزایش می‌دهیم. برای مقدار ۵ شمارنده‌های $c_{۱}^{۳}$ و $c_{۲}^{۴}$ را کاهش و $c_{۳}^{۴}$ را افزایش می‌دهیم. برای مقدار ۶ شمارنده‌های $c_{۱}^{۴}$ و $c_{۳}^{۳}$ را کاهش و $c_{۲}^{۱}$ را افزایش می‌دهیم. ساختار نهایی به شکل زیر است:

	۱	۲	۳	۴	۵
$h_۱$	−۶	۰	۹	−۱	۰
$h_۲$	۱	۳	۱	−۱	۰
$h_۳$	−۷	۰	−۴	۷	۰

روش معمول برای تولید تقریب‌های بهتر از تعدادی آزمایش توزیع شده تصادفی در نظریه احتمال استفاده از میانگین و میانه است. الگوریتم طرحواره شمارش از میانه به دلیل ثبات و حساسیت کمتر نسبت به نقاط پرت در محاسبه تخمین نهایی بسامد استفاده می‌کند.

الگوریتم ۶: تخمین بسامد با طرحواره شمارش
ورودی: مقدار $x \in D$
ورودی: طرحواره شمارش با شمارنده‌های $p \times m$
خروجی: تخمین بسامد
$\hat{f} = \{\hat{f}_j\}_{i=۱}^{p}$
برای $j \leftarrow ۱$ تا p انجام بده
$i \leftarrow h_j(x)$
$\hat{f}_j \leftarrow s_j(x) \cdot c_j^i$
برگرداندن میانه $(\hat{f}_۱, \hat{f}_۲, \ldots, \hat{f}_p)$

زمان به‌روزرسانی برای هر مقدار O(p) است و برای یافتن میانه p عنصر، مقداری زمان خطی با استفاده از یکی از الگوریتم‌های انتخاب صرف می‌کنیم، بنابراین زمان کلی پرس‌وجو O(p) است.

مثال ۶- تخمین بسامد با طرحواره شمارش: داده‌ساختاری مثال را در نظر می‌گیریم:

	۱	۲	۳	۴	۵
$h_۱$	−۶	۰	۹	−۱	۰
$h_۲$	۱	۳	۱	−۱	۰
$h_۳$	−۷	۰	−۴	۷	۰

بسامد مقدار ۴ را تخمین می‌زنیم که شمارنده‌های متناظر آن c_1^3، c_2^3 و c_3^1 هستند، جهت‌های به‌روزرسانی $s_1(4) = 1$، $s_2(4) = 1$ و $s_3(4) = -1$ هستند، همانطور که قبلا تعیین کردیم. با استفاده از الگوریتم ۶، به عنوان تخمین، میانه مقادیر وزنی آن شمارنده‌ها را محاسبه می‌کنیم:

$$\hat{f} = median(s_1(4) \cdot c_1^3, s_2(4) \cdot c_2^3, s_3(4) \cdot c_3^1) = median(9,1,7) = 7.$$

بنابراین، بسامد تخمینی مقدار ۴ برابر با ۷ است که همان شمارش صحیح از مجموعه داده است.

حال، مقدار ۲ را با شمارنده‌های متناظر c_1^3، c_2^2 و c_3^3 با مقادیر توابع درهم‌ساز جهت به‌روزرسانی $s_1(2) = 1$، $s_2(2) = 1$ و $s_3(2) = -1$ در نظر می‌گیریم. بنابراین، تخمین بسامد برای مقدار ۲ است

$$\hat{f} = median(s_1(2) \cdot c_1^3, s_2(2) \cdot c_2^2, s_3(2) \cdot c_3^3) = median(9,3,4) = 4,$$

که مقدار واقعی ۳ را بیش از حد تخمین می‌زند.

الگوریتم طرحواره شمارش را می‌توان برای یافتن k مقدار پربسامد، که به عنوان مسئله پربسامد شناخته می‌شود، استفاده کرد. در یک گذر از جریان داده، و همچنین آرایه منظم $p \times m$ از شمارنده‌ها و توابع درهم‌ساز $\{h_j\}_{j=1}^{p}$ و $\{s_j\}_{j=1}^{p}$، مجموعه‌ای از X^* از k مقدار با بسامد بالا را حفظ می‌کنیم. ابتدا هر مقدار x را از جریان داده به داده‌ساختار طبق الگوریتم ۵ نمایه می‌شود. سپس، اگر مقدار در مجموعه X^* نباشد و ظرفیت اضافه کردن آن وجود داشته باشد، مقدار را درج می‌کنیم. در غیر این صورت، بسامد را با الگوریتم ۶ تخمین می‌زنیم و اگر بیشتر از کوچکترین بسامد در مجموعه باشد، مقدار x را به X^* اضافه می‌کنیم و همزمان مقدار با کوچکترین بسامد را حذف می‌کنیم.

الگوریتم ۷- دریافت مقادیر پربسامد با طرحواره شمارش
ورودی: جریان داده D
ورودی: طرحواره شمارش با شمارنده‌های $p \times m$
خروجی: مقادیر پربسامد
$X^* \leftarrow \emptyset$
برای $x \in D$ انجام بده
Update(x)
اگر $x \in X^*$ آنگاه
continue
اگر $|X^*| < k$ آنگاه
$X^* \leftarrow X^* \cup \{x\}$
در غیر این صورت
$\hat{f} \leftarrow Frequency(x)$
$(x^*_{min}, \hat{f}^*_{min}) \leftarrow \min_{x^* \in X^*} Frequency(x^*)$
اگر $\hat{f} > \hat{f}^*_{min}$ آنگاه
$X^* \leftarrow X^* \cup \{x\} \backslash \{x^*_{min}\}$
برگرداندن X^*

مثال ۷- پربسامدترین مقادیر با طرحواره شمارش
مجموعه داده مثال‌های قبلی را در نظر می‌گیریم و k = ۳ مقدار پربسامد در مجموعه داده جستجو می‌کنیم:
{۴,۴,۴,۴,۲,۳,۵,۴,۶,۴,۳,۳,۴,۲,۳,۳,۳,۲}

با توجه به الگوریتم ۷، علاوه بر داده‌ساختار مجموعه X^* برای ذخیره نامزدهای پربسامد ایجاد می‌کنیم. مقادیر مجموعه داده را به ترتیب می‌خوانیم که مقدار ۴ است، بنابراین، همانطور که از مثال ۵ می‌دانیم، باید شمارنده‌های c_1^3 و c_2^3 را افزایش و شمارنده c_3^1 را کاهش دهیم.

	۱	۲	۳	۴	۵
h_1	۰	۰	۱	۰	۰
h_2	۰	۰	۱	۰	۰
h_3	−۱	۰	۰	۰	۰

مجموعه X^* تهی است، بنابراین مقدار ۴ را در آن درج می‌کنیم: $X^* = [4]$. سه مقدار بعدی در جریان داده نیز برابر با ۴ هستند، در نتیجه، آنها بدون تغییر در X^* در داده‌ساختار نمایه می‌شوند.

	۱	۲	۳	۴	۵
h_1	۰	۰	۴	۰	۰
h_2	۰	۰	۴	۰	۰
h_3	−۴	۰	۰	۰	۰

مقدار بعدی ۲ است و همانطور که قبلا نشان دادیم، شمارنده‌های c_1^3 و c_2^2 را افزایش و c_3^3 را کاهش می‌دهیم. این مقدار در مجموعه پربسامدترین نامزدها نیست و از آنجایی که X^* ظرفیت کافی دارد، مقدار ۲ را به مجموعه اضافه می‌کنیم: $X^* = [4,2]$.

	۱	۲	۳	۴	۵
h_1	۰	۰	۵	۰	۰
h_2	۰	۱	۴	۰	۰
h_3	−۴	۰	−۱	۰	۰

مقدار ورودی بعدی ۳ است. برای اندیس کردن آن در داده‌ساختار، شمارنده‌های c_1^1 و c_2^3 را کاهش و شمارنده c_3^4 را افزایش می‌دهیم. چون مجموعه X^* صرفا شامل دو مقدار از سه مقدار ممکن است، مقدار ۳ را به مجموعه اضافه می‌کنیم: $X^* = [4,2,3]$.

	۱	۲	۳	۴	۵
h_1	−۱	۰	۵	۰	۰
h_2	۰	۱	۳	۰	۰
h_3	−۴	۰	−۱	۱	۰

سپس، مقدار ۵ را از مجموعه داده گرفته و در طرحواره شمارنده‌های c_1^3 و c_2^4 را کاهش و شمارنده c_3^4 را افزایش می‌دهیم.

	۱	۲	۳	۴	۵
h_1	−۱	۰	۴	۰	۰
h_2	۰	۱	۳	−۱	۰
h_3	−۴	۰	−۱	۲	۰

مقدار ۵ در مجموعه X^* نیست که به حداکثر ظرفیت $k = ۳$ مقدار تحت نظر رسیده است. بنابراین، باید بسامدهای مقادیر موجود در مجموعه و مقدار ۵ را با استفاده از الگوریتم ۶ برای داده‌ساختار فعلی تخمین بزنیم.

$$\hat{f}(۵) = median(-c_1^3, -c_2^4, c_3^4) = median(-۴,۱,۲) = ۱,$$

$$\hat{f}(۴) = median(c_۱^۳, c_۲^۳, -c_۳^۱) = median(۴,۳,۴) = ۴,$$
$$\hat{f}(۲) = median(c_۱^۳, c_۲^۲, -c_۳^۳) = median(۴,۱,۱) = ۱,$$
$$\hat{f}(۳) = median(-c_۱^۱, -c_۲^۳, c_۳^۴) = median(۱, -۳,۲) = ۱.$$

بنابراین، بسامد تخمینی مقدار فعلی ۵ از حداقل بسامد مقادیر موجود در مجموعه تجاوز نمی‌کند، در نتیجه، مجموعه مقادیر تحت نظر را تغییر نمی‌دهیم: $X^* = [۴,۲,۳]$. به روشی مشابه، همه مقادیر باقی‌مانده را از مجموعه داده مدیریت می‌کنیم و پس از پردازش آخرین عنصر، داده‌ساختار به شکل زیر است:

	۱	۲	۳	۴	۵
$h_۱$	−۶	۰	۹	−۱	۰
$h_۲$	۱	۳	۱	−۱	۰
$h_۳$	−۷	۰	−۴	۷	۰

و مجموعه سه مقدار پربسامد عبارت است از $X^* = [۴,۲,۳]$. شایان توجه است پربسامدترین مقادیر در X^* مرتب نیستند و برای تخمین بسامد آنها می‌توان از الگوریتم ۶ استفاده کرد.

به همین ترتیب می‌توان به مسئله مقادیر پربسامد بپردازیم. برای یافتن k مقدار پربسامد، شمارندهٔ N از مقادیر پردازش شده را تعریف می‌کنیم، تا با استفاده از آن، آستانه بسامد $f^* = \frac{N}{k}$ را هر بار که مقداری جدید نمایه می‌شود محاسبه می‌کنیم. اگر بسامد تخمینی مقدار فعلی بالاتر از آستانه باشد، آن را به عنوان نامزد مقادیر پربسامد در هرم X^* درج می‌کنیم. علاوه بر این، در هر مرحله عناصری را از هرم حذف می‌کنیم که بسامد ذخیره شده آنها کمتر از آستانه فعلی f^* می‌افتد.

الگوریتم ۸- تعیین مقادیر پربسامد با طرحواره شمارش

ورودی: جریان داده D

ورودی: طرحواره شمارش با $p \times m$ شمارنده

خروجی: مقادیر پربسامد

$N \leftarrow ۰, X^* \leftarrow \emptyset$

برای $x \in D$ انجام بده

$N \leftarrow N + ۱$

Update(x)

$\hat{f} \leftarrow Frequency(x)$

$f^* \leftarrow \frac{N}{k}$

اگر $\hat{f} \geq f^*$ آنگاه

$X^* \leftarrow X^* \cup \{(x, \hat{f})\}$

برای $(x^*, \hat{f}) \in X^*$ انجام بده

اگر $\hat{f} \leq f^*$ آنگاه

$X^* \leftarrow X^* \setminus \{(x^*, \hat{f})\}$

برگرداندن X^*

الگوریتم طرحواره شمارش را می‌توان برای یافتن مقادیر با بیشترین تغییر بسامد، که به عنوان مسئله حداکثر تغییر نیز شناخته می‌شود، به کار برد. با داشتن جریان‌های داده از دو دوره قابل مقایسه، می‌توانیم داده‌ساختاری برای هر یک از آنها بسازیم و هرم X^* از مقادیر با بزرگترین تفاوت‌ها را حفظ کنیم. هر بار که مقادیر جدید نمایه می‌شوند، بسامد آنها را با استفاده از الگوریتم ۶ تخمین می‌زنیم و هرم را به‌روزرسانی می‌کنیم تا فقط مقادیر با بیشترین تغییر را نگه داریم. در نهایت، الگوریتم k مقدار با بزرگترین مقادیر تغییر بسامد را خروجی می‌دهد.

الگوریتم ۹- تعیین تغییر بیش با طرحواره شمارش

ورودی: جریان داده قدیم D و جدید D

ورودی: دو طرحواره شمارش با $p \times m$ شمارنده قدیم و جدید

خروجی: مقادیر پربسامد

$N_{\text{قدیم}} \leftarrow 0, X^*_{\text{قدیم}} \leftarrow \emptyset$

برای قدیم $x \in D$ انجام بده

$N_{\text{قدیم}}++$

Update(x)

$\hat{f} \leftarrow Frequency(x)$

$f^* \leftarrow \frac{N_{\text{قدیم}}}{k}$

اگر $\hat{f} \geq f^*$ آنگاه

$X^*_{\text{قدیم}} \leftarrow X^*_{\text{قدیم}} \cup \{(x, \hat{f})\}$

برای قدیم $(x^*, \hat{f}) \in X^*_{\text{قدیم}}$ انجام بده

اگر $\hat{f} \leq f^*$ آنگاه

$X^*_{\text{قدیم}} \leftarrow X^*_{\text{قدیم}} \setminus \{(x^*, \hat{f})\}$

$N_{\text{جدید}} \leftarrow 0, X^*_{\text{جدید}} \leftarrow \emptyset$

برای جدید $x \in D$ انجام بده

$N_{\text{جدید}}++$

Update(x)

$\hat{f} \leftarrow Frequency(x)$

$f^* \leftarrow \frac{N_{\text{جدید}}}{k}$

اگر $\hat{f} \geq f^*$ آنگاه

$X^*_{\text{جدید}} \leftarrow X^*_{\text{جدید}} \cup \{(x, \hat{f})\}$

برای جدید $(x^*, \hat{f}) \in X^*_{\text{جدید}}$ انجام بده

اگر $\hat{f} \leq f^*$ آنگاه

$X^*_{\text{جدید}} \leftarrow X^*_{\text{جدید}} \setminus \{(x^*, \hat{f})\}$

$X^* = X^*_{\text{قدیم}} \cup X^*_{\text{جدید}}$

$T \leftarrow \phi$

برای $x \in X^*$

tمحاسبهٔ اختلاف مقدار بسامد جدید و قدیم.

$T = T \cup \{t\}$

مرتب‌سازی نزولی |T|

برگرداندن k مقدار اول T بر اساس قدر مطلق آن

ویژگی‌ها

طرحواره شمارش تضمین می‌کند که خطای تخمین برای بسامدها با احتمال حداقل $1 - \delta$ بیشتر از $\epsilon \cdot n$ نباشد. افزایش تعداد توابع درهم‌ساز p احتمال تخمین بد را کاهش می‌دهد و برای خطای استاندارد مطلوب δ توصیه در مورد تعداد توابع درهم‌ساز، که مربوط به ردیف‌های داده‌ساختار است، عبارت است از

$$p = \left\lceil \ln \frac{1}{\delta} \right\rceil.$$

هرچه m بزرگتر باشد، احتمال وقوع تصادم کمتر است، به این معنی که خطای تخمین $\epsilon \cdot n$ کمتر خواهد بود. در عین حال، با p بزرگتر، تخمین‌گرهای بیشتری برای محاسبه مقدار نهایی استفاده می‌شوند که آن را قابل اعتمادتر می‌کند. توصیه در مورد تعداد شمارنده‌ها m عبارت است از

$$m \approx \left\lceil \frac{2.71828}{\epsilon^2} \right\rceil$$

کل فضای مورد نیاز داده‌ساختار $O(m \cdot p + 2p)$ است. زیرا یک ماتریس شمارش به اندازه $p \times m$ و دو تابع درهم‌ساز به ازای هر ردیف نگه می‌داریم. اگر دو داده‌ساختار دارای اندازه m یکسانی باشند، می‌توان آنها را به راحتی به یکدیگر اضافه و از یکدیگر کم

کرد، این برای پردازش جریان توزیع شده مفید است. پیاده‌سازی‌هایی از برای Apache Hive و سایر نرم‌افزارهای انبار داده وجود دارد، اما برنامه‌های جدید تمایل دارند از جانشین آن، الگوریتم طرحواره شمارش-کمینه، به دلیل نیاز به فضای کمتر و زمان اجرای کمتر استفاده کنند.

طرحواره شمارش-کمینه

طرحواره شمارش-کمینه داده‌ساختار احتمالاتی ساده و فضا-کارآمد است که برای تخمین بسامد مقادیر در جریان‌های داده استفاده می‌شود و می‌تواند به مسئله مقادیر پربسامد بپردازد. در سال ۱۳۸۲ توسط گراهام کورمودی و شان موتوکریشنان معرفی و در سال ۱۳۸۴ منتشر شد.

همانطور که در بخش قبل در طرحواره شمارش دیدیم، مانع اصلی کاربرد مستقیم فیلتر بلوم شمارنده در تخمین بسامد اشتراک‌گذاری آرایه‌ای واحد از شمارنده‌ها برای همه توابع درهم‌ساز است و در نتیجه دچار تصادم‌های سخت و نرم است.

کیفیت تخمینْ سخت تحت تأثیر احتمال تصادم‌های درهم‌ساز است. با این حال، هنگامی که تعداد مقادیر در جریان داده عظیم است، تصادم با مقادیر با بسامد بالا تقریبا قطعی است و موجب بی‌فایدگی چنین تقریبی به دلیل بیش‌تخمینی بزرگ همه شمارنده‌ها می‌شود.

الگوریتم طرحواره شمارش-کمینه جهت حل مشکل تخمین‌های بد، آرایهٔ واحد m شمارنده را با جدول درهم‌ساز از p آرایه از m شمارنده جانشین می‌کند و به جای به‌روزرسانی هر شمارنده توسط هر عنصر، اجازه می‌دهد مقادیر زیرمجموعه‌های مختلفی از شمارنده‌ها را به‌روزرسانی کنند. هدف m فشرده‌سازی جریان داده $D = \{x_1, x_2, \dots, x_n\}$ است و به دلیل $m \ll n$، فشرده‌سازی «با اتلاف» است که منجر به خطا می‌شود. برای کاهش این خطاها، الگوریتم با استفاده از p تابع درهم‌ساز با آرایه اختصاصی از m شمارنده برای هر کدام، آزمایش‌های مستقل زیادی را ترتیب می‌دهد. داده‌ساختار فضا-کارآمد از آرایه $p \times m$ از شمارنده‌ها $\{c_j^i\}$ تشکیل شده است، که در آن p تابع درهم‌ساز مستقل از هم $h_1, h_2, \dots, h_p$ مقادیر دامنهٔ ورودی را بازهٔ $\{1,2, \dots, m\}$ نگاشت می‌کنند. چنین داده‌ساختاری امکان نمایه‌سازی مقادیر جریان داده را فراهم می‌کند، و منجر به به‌روزرسانی شمارنده‌ها می‌شود و می‌تواند تعداد دفعاتی که هر مقدار خاص نمایه شده است را حساب کند. این مقدار به عنوان تخمین بسامد مقدار به کار می‌رود.

الگوریتم ۱۰- به‌روزرسانی طرحواره شمارش-کمینه
ورودی: مقدار $x \in D$
ورودی: طرحواره شمارش-کمینه با $p \times m$ شمارنده
برای $j \leftarrow 1$ تا p انجام بده
$i \leftarrow h_j(x)$
$c_j^i \leftarrow c_j^i + 1$

با فرض اینکه هر تابع درهم‌ساز $\{h_j\}_{j=1}^{p}$ را می‌توان در زمان ثابت محاسبه کرد، زمان اجرای روش به‌روزرسانی الگوریتم ۹ O(p) است.

مثال ۸- ساختن طرحواره شمارش-کمینه

مجموعه داده با $n = ۱۸$ مقدار را بار دیگر در نظر می‌گیریم:

$$\{۴,۴,۴,۴,۲,۳,۵,۴,۶,۴,۳,۳,۴,۲,۳,۳,۳,۲\}$$

طرحواره شمارش-کمینه از $m = ٤$ شمارنده با استفاده از $p = ۲$ تابع درهم‌ساز مبتنی بر مورمور۳ و فـن‌و۱الف را ایجاد می‌کنیم:

$$h_۱(x) = MurmurHash3(x)\%\ ۴ + ۱,$$
$$h_۲(x) = FNV1a(x)\%\ ۴ + ۱$$

در ابتدا، داده‌ساختار از صفر تشکیل شده است:

	۱	۲	۳	۴
$h_۱$	۰	۰	۰	۰
$h_۲$	۰	۰	۰	۰

مقادیر ورودی را یک به یک می‌خوانیم. اولین مقدار ۴ است و با توجه به الگوریتم ۱۰، مقادیر درهم آن را برای تعیین شمارنده‌هایی که باید به‌روزرسانی شوند محاسبه می‌کنیم:

$$i_۱ = h_۱(۴) = ۴,$$
$$i_۲ = h_۲(۴) = ۴$$

با وجود اینکه هر دو تابع خروجی یکسانی دارند، به دلیل خیره در شمارنده‌های متفاوت تصادمی رخ نمی‌دهد و مشکلی پیش نمی‌آید. بنابراین، شمارنده‌های $c_۱^۴$ و $c_۲^۴$ را افزایش می‌دهیم و ساختار به صورت زیر در می‌آید.

	۱	۲	۳	۴
$h_۱$	۰	۰	۰	۱
$h_۲$	۰	۰	۰	۱

سه مقدار بعدی همگی برابر با ۴ هستند، بنابراین همان شمارنده‌ها را به‌روزرسانی می‌کنیم:

	۱	۲	۳	۴
$h_۱$	۰	۰	۰	۴
$h_۲$	۰	۰	۰	۴

مقدار بعدی در مجموعه داده ۲ است و اندیس‌های متناظر آن $i_۱ = ۴$ و $i_۲ = ۱$ هستند، بنابراین شمارنده‌های $c_۱^۴$ و $c_۲^۱$ را افزایش می‌دهیم. توجه داشته باشید که تصادمی نرم وجود دارد و مقدار ۲ شمارنده‌ای را که مقدار ۴ استفاده می‌کند به‌روزرسانی می‌کند. در نتیجه، مقدار موجود در شمارنده $c_۱^۴$ مقدار واقعی را برای هر دو مقدار بیش از حد تخمین می‌زند.

	۱	۲	۳	۴
$h_۱$	۰	۰	۰	۵
$h_۲$	۱	۰	۰	۴

به همین ترتیب، همه مقادیر باقی‌مانده را پردازش می‌کنیم و شمارنده‌های $c_۱^۱$ و $c_۲^۳$ را برای مقدار ۳، شمارنده‌های $c_۱^۱$ و $c_۲^۱$ را برای مقدار ۵، و $c_۱^۱$ و $c_۲^۲$ را برای مقدار ۶ به‌روزرسانی می‌کنیم. لازم به ذکر است هر دو شمارنده برای مقدار ۶ با مقادیر دیگر تصادم می‌کنند، بنابراین می‌توانیم انتظار داشته باشیم که مقدار آن بیش از حد تخمین زده شود. داده‌ساختار نهایی به شکل زیر است:

	۱	۲	۳	۴
$h_۱$	۸	۰	۰	۱۰
$h_۲$	۱	۴	۶	۷

با هر بار نمایه شدن مقدار x، شمارنده‌های $c_j^{h_j(x)}$ برای هر ردیف j از طرحواره افزایش می‌یابند و کاهش در کار نیست. در نتیجه، شمارنده‌ها کران بالای بسامدها را فراهم می‌کنند:

$$f(x) \leq c_j^{h_{j(x)}}, j = ۱,۲, \dots, p$$

چون $m \ll n$ تصادم‌های زیادی وجود دارد، به طوری که ممکن است برای $x \neq y$ ، $h_j(x) = h_j(y)$. در نتیجه، هنگامی مقدار y در نمایه شود، شمارندهٔ مقدار x نیز افزایش می‌یابد. پس، شمارنده‌ها قادر به تخمین کمتر از حد بسامد واقعی f(x) نیستند، و معمولا بسامد بیش از حد تخمین می‌زنند. در نتیجه، p تخمین وجود دارد که از خطای یک‌طرفه رنج می‌برند (همه آنها بیش‌تخمینی از مقدار واقعی هستند). روش معمول برای ساختن تقریب بهتر از تعدادی تخمین، میانگین‌گیری است، اما این خطا می‌تواند تخمین را بدتر نیز کند. سخن کوتاه، بهترین تخمین در این مورد، کوچکترین آنهاست.

الگوریتم ۱۱- تخمین بسامد با طرحواره شمارش-کمینه
ورودی: مقدار $x \in D$
ورودی: طرحواره شمارش-کمینه با شمارنده‌های $p \times m$
خروجی: تخمین بسامد
$\hat{f} = \{\hat{f}_j\}_{i=1}^{p}$
برای j از ۱ تا p انجام بده
$i \leftarrow h_j(x)$
$\hat{f}_j \leftarrow c_j^i$
برگرداندن $min\left(\hat{f}_1, \hat{f}_2, \dots, \hat{f}_p\right)$

حداقل p مقدار را می‌توان در زمان خطی پیدا کرد، و بنابراین زمان اجرای روش تخمین بسامد که توسط الگوریتم ۱۱ همانند به‌روزرسانی O(p) است.

مثال ۱۰- تخمین بسامد با طرحواره شمارش-کمینه
داده‌ساختار CMSKETCH را که در مثال ۴.۸ ساختیم در نظر می‌گیریم:

	۱	۲	۳	۴
h_1	۸	۰	۰	۱۰
h_2	۱	۴	۶	۷

بسامد مقدار ۴ را تخمین می‌زنیم که شمارنده‌های متناظر آن c_1^4 و c_2^4 هستند. با استفاده از الگوریتم ۱۱، به عنوان تخمین، حداقل آن شمارنده‌ها را محاسبه می‌کنیم:

$$\hat{f} = min\,(c_1^4, c_2^4) = min\,(10,7) = 7$$

بنابراین، بسامد تخمینی مقدار ۴ برابر با ۷ است که همان شمارش صحیح از مجموعه داده است. حال، مقدار ۶ را با شمارنده‌های متناظر c_1^1 و c_2^2 در نظر می‌گیریم. با این حال، همانطور که قبلا اشاره کردیم، هر دو آنها به دلیل تصادم ناشی از مقادیر دیگر نیز استفاده می‌شوند. بنابراین، تخمین بسامد برای مقدار ۶ است

$$\hat{f} = min\,(c_1^1, c_2^2) = min\,(8,4) = 4,$$

که مقدار واقعی ۱ را به طور قابل توجهی بیش از حد تخمین می‌زند. اگر بخواهیم دقت بهتری حفظ کنیم و چنین تصادم‌هایی را نادر کنیم، به توابع درهم‌ساز و شمارنده‌های بیشتری نیاز داریم که زمان محاسباتی و ذخیره‌سازی را افزایش می‌دهد.

با دانستن نحوه تخمین بسامد عناصر، الگوریتم طرحواره شمارش-کمینه اجازه می‌دهد پربسامدترین مقادیر تعیین شوند. مشابه طرحواره شمارش، ساده‌ترین رویکرد مستلزم حفظ مجموعه‌ای از نامزدهای پربسامدترین مقادیر و همچنین داده‌ساختار اصلی است. سپس، از جریان داده را می‌خواند و طرحواره با تمام مقادیری که تاکنون دیده شده‌اند به‌روزرسانی می‌شود. اگر مقدار در مجموعه

تحت نظر نباشد و ظرفیت مجموعه نیز پر نباشد، آن را اضافه می‌کنیم. با این حال، اگر مجموعه در حداکثر ظرفیت خود باشد، مقدار فعلی را تنها در صورتی اضافه می‌کنیم که بسامد تخمینی آن از حداقل بسامد موجود در مجموعه بیشتر باشد و مقدار با کوچکترین بسامد را جایگزین کنیم. در پایان، مقادیر موجود در X^* به عنوان پربسامدترین مقادیر در جریان داده در نظر گرفته می‌شوند.

به روشی مشابه، داده‌ساختار می‌تواند به مسئله مقادیر پربسامد را حل کند. در یک گذر از جریان داده، علاوه بر آرایه $p \times m$ از شمارنده‌ها C و p تابع درهم‌ساز، شمارنده N را اختصاص می‌دهیم که تعداد مقادیر دیده شده تاکنون را ذخیره می‌کند و هرم X^* از حداکثر k مقدار پربسامد بالقوه را پشتیبانی می‌کند. از آستانه بسامد $f^* = \frac{N}{k}$ برای تصمیم‌گیری در مورد پربسامد بودن مقدار استفاده می‌شود. برای هر مقدار x در جریان داده، رویهٔ به‌روزرسانی و سپس تخمین بسامد را اجرا می‌کنیم. اگر $\hat{f}(x) \geq f^*$ باشد، آن‌گاه مقدار به عنوان نامزد مقدار پربسامد واجد شرایط است. اگر مقدار هنوز در هرم نیست، آن را همراه با بسامد آن ذخیره، و در صورت وجود بسامد آن را افزایش می‌دهیم.

شمارنده N با هر مقدار پردازش شده افزایش می‌یابد. زمانی که N افزایش می‌یابد، بسامد تخمینی برای برخی از مقادیر در هرم کمتر از f^* می‌شود و این مقادیر باید از هرم حذف شوند. در پایان پردازش، همه مقادیر موجود در هرم به عنوان مقادیر پربسامد در نظر گرفته می‌شوند. طبق تعریف، حداکثر k مقدار پربسامد در جریان داده وجود دارد.

الگوریتم ۱۲- مقادیر پربسامد با طرحواره شمارش-کمینه
ورودی: جریان داده D
ورودی: طرحواره شمارش-کمینه با شمارنده‌های $p \times m$
خروجی: مقادیر پربسامد
$N \leftarrow ۰, X^* \leftarrow \emptyset$
برای $x \in D$ انجام بده
$N \leftarrow N + ۱$
Update(x)
$\hat{f} \leftarrow Frequency(x)$
$f^* \leftarrow \frac{N}{k}$
اگر $\hat{f} \geq f^*$ آنگاه
$X^* \leftarrow X^* \cup (x, \hat{f})$
برای $(x^*, \hat{f}) \in X^*$ انجام بده
اگر $\hat{f} \leq f^*$ آنگاه
$X^* \leftarrow X^* \setminus (x^*, \hat{f})$
برگرداندن X^*

نگهداری هرم برای مسئله ϵ-مقادیر پربسامد با $\epsilon = \frac{۱}{۲k}$ نیاز به کار اضافی $O\left(\log\frac{۱}{\epsilon}\right)$ به ازای هر مقدار دارد.

مثال ۱۰- مقادیر پربسامد با طرحواره شمارش-کمینه

همان تنظیمات مثال ۴.۹ را در نظر می‌گیریم و به دنبال k = ۳ مقدار پربسامد در حین پردازش مجموعه داده بگردید.
{۴,۴,۴,۴,۲,۳,۵,۴,۶,۴,۳,۳,۴,۲,۳,۳,۳,۲}

با استفاده از الگوریتم۱۱، علاوه بر داده‌ساختار ، شمارنده N از مقادیر پردازش شده و هرم X^* که حداکثر k نامزد مقادیر پربسامد را ذخیره می‌کند، ایجاد می‌کنیم. جزئیات به‌روزرسانی شمارنده‌ها و تخمین بسامد را نادیده می‌گیریم زیرا این مراحل مانند مثال‌های بالا هستند. شروع به مصرف مجموعه داده می‌کنیم و اولین مقدار ۴ است، بنابراین شمارنده‌های مربوطه c_1^4 و c_2^4 را افزایش می‌دهیم.

	۱	۲	۳	۴
h_1	۰	۰	۰	۱
h_2	۰	۰	۰	۱

در این مرحله، N = ۱ مقدار را پردازش کرده‌ایم، بنابراین آستانه f^* برای هرم X^* برابر با $\frac{۱}{۳}$ است. تخمین بسامد مقدار ۴ از برابر ۱ است که بالاتر از آستانه است، بنابراین این مقدار و بسامد آن را به هرم اضافه می‌کنیم: $X^* = [(۴,۱)]$. مقدار بعدی دوباره ۴ است و همان شمارنده‌ها را افزایش می‌دهیم.

	۱	۲	۳	۴
h_1	۰	۰	۰	۲
h_2	۰	۰	۰	۲

N = ۲ مقدار پردازش شده است، آستانه f^* به $\frac{۲}{۳}$ تغییر می‌کند. تخمین بسامد فعلی برای مقدار ۴ برابر ۲ است که هنوز بالاتر از آستانه است و به دلیل وجود از قبل مقدار در هرم، فقط بسامد آن را به‌روزرسانی می‌کنیم: $X^* = [(۴,۲)]$. به روشی مشابه، ۱۴ مقدار بعدی را پردازش می‌کنیم (تا N = ۱۶). هیچ تغییری در تعداد مقادیر هرم وجود ندارد و مقدار ۴ تنها نامزد مقدار پربسامد تاکنون است: $X^* = [(۴,۷)]$. داده‌ساختار به شکل زیر است:

	۱	۲	۳	۴
h_1	۶	۰	۰	۹
h_2	۱	۳	۵	۷

مقدار بعدی در مجموعه داده ۳ است که شمارنده‌های c_1^1 و c_2^3 آن را افزایش می‌دهیم.

	۱	۲	۳	۴
h_1	۷	۰	۰	۹
h_2	۱	۳	۶	۷

در این لحظه، N = ۱۷ مقدار پردازش شده است، بنابراین آستانه بسامد $f^* = \frac{۱۷}{۳} \approx ۵.۳۳$ است. بسامد تخمینی مقدار ۳ برابر $\hat{f} = min\,(۷,۶) = ۶$ است که بالاتر از آستانه است، بنابراین آن را به هرم اضافه می‌کنیم: $X^* = [(۴,۷), (۳,۶)]$. همه مقادیر در هرم بسامدهای به اندازه کافی بزرگ دارند، بنابراین هیچ یک از آنها را حذف نمی‌کنیم.

آخرین مقدار در مجموعه داده ۲ است که بسامد آن کمتر از آستانه $f^* = \frac{۱۸}{۳} = ۶$ است. بنابراین، در این مرحله هیچ تغییری در هرم وجود ندارد و فهرست نهایی مقادیر پربسامد برابر است با $X^* = [(۴,۷), (۳,۶)]$

ویژگی‌ها

طرحواره شمارش-کمینه روشی تقریبی - احتمالاتی است، بنابراین دو پارامتر، خطای ϵ در پاسخ به پرس‌وجوی خاص و احتمال خطای δ، بر نیازهای فضا و زمان تأثیر می‌گذارند. به بیان دیگر، روش تضمین می‌کند که خطای تخمین بسامدها با احتمال حداقل $۱ - \delta$ از $\epsilon \cdot n$ تجاوز نخواهد کرد.

مشابه طرحواره شمارش، افزایش تعداد توابع درهم‌ساز p احتمال تخمین بد را کاهش می‌دهد. برای خطای استاندارد مطلوب δ، توصیه برای تعداد توابع درهم‌ساز که مربوط به ردیف‌های داده‌ساختار هستند عبارت است از

$$p = \lceil \ln ۱/\delta \rceil .$$

هرچه m بزرگتر باشد، احتمال وقوع تصادم کمتر است، بنابراین خطای بیش‌تخمینی $\epsilon \cdot n$ کمتر خواهد بود. در عین حال، با p بزرگتر، تخمین‌های بیشتری برای محاسبه مقدار حداقل نهایی استفاده می‌شوند که آن را قابل اعتمادتر می‌کند. بنابراین، توصیه در مورد تعداد شمارنده‌ها m عبارت است از

$$m \approx \left\lceil \frac{۲.۷۱۸۲۸}{\epsilon} \right\rceil,$$

که در مقایسه با مقدار موردنیاز با طرحواره شمارش نشان می‌دهد که طرحواره شمارش-کمینه نسبت به طرحواره شمارش فضا کمتری لازم دارد. چون داده‌ساختار از آرایه دو بعدی به اندازه $p \times m$ تشکیل و از p تابع درهم‌ساز استفاده می‌کند، به فضای O(mp + p) نیاز دارد.

مثال ۴.۱۱: تخمین فضای مورد نیاز
با توجه به شرط بالا، برای داشتن خطای استاندارد δ حدود یک درصد، حداقل $p = \left\lceil \ln \frac{۱}{۰.۰۱} \right\rceil = ۵$ تابع درهم‌ساز مورد نیاز است. برای مثال، انتظار داریم ۱۰ میلیون ($n = ۱۰^۷$) مقدار نمایه شوند و اجازه می‌دهیم بیش‌تخمینی ثابت ۱۰ داشته باشیم. بنابراین، به $\epsilon = \frac{۱۰}{۱۰^۷} = ۱۰^{-۶}$ نیاز داریم و تعداد توصیه شده شمارنده‌ها است

$$m = \frac{۲.۷۱۸۲۸}{۱۰^{-۶}} \approx ۲۷۱۸۲۸۰$$

بنابراین، داده‌ساختار باید آرایه شمارنده‌ای به اندازه ۲۷۱۸۲۸۰ × ۵ نگه دارد و با داشتن شمارنده‌های صحیح ۳۲ بیتی، کل داده‌ساختار به ۵۴.۴ مگابایت حافظه نیاز دارد.

دو طرحواره شمارش-کمینه با اندازه یکسان را می‌توان به راحتی با جمع ماتریسی ساده با هم ادغام کرد و در نتیجه داده‌ساختاری برای اتحاد مجموعه‌های داده آنها به دست آورد. این قابلیت موجب می‌شود استفاده ای طرحواره شمارش-کمینه در MapReduce و جریانی موازی برنامه‌های داده بزرگ مفید واقع شود.

داده بزرگ دارای چند مشخصه است. چنین داده‌های دارای حجم زیادی هستند با سرعت و تعداد زیاد وارد می‌شوند، در نتیجه فضا و زمان به‌روزرسانی مناسب را طلب می‌کند. پیاده‌سازی‌های عملی طرحواره شمارش-کمینه فقط تا چند صد مگابایت حافظه مصرف می‌کنند و می‌توانند ده‌ها میلیون به‌روزرسانی در ثانیه را مدیریت کنند.

طرحواره شمارش-کمینه به طور گسترده برای وظایف تحلیل ترافیک و برنامه‌های استخراج درون‌جریانی که بر روی چارچوب‌های پردازش جریان توزیع شده مانند Apache Spark، Apache Storm، Apache Flink و دیگران اجرا می‌شوند، استفاده می‌شود. همچنین پیاده‌سازی‌هایی برای پایگاه‌های داده محبوب مانند Redis و PostgreSQL وجود دارد.