الگوریتم‌های احتمالی عضویت

محسن هوشمند

فهرست

عضویت

مسئله عضویت در مجموعه داده به عمل تصمیم‌گیری در مورد تعلق یا عدم تعلق مقداری به آن مجموعه اطلاق می‌شود. عضویت در مجموعه‌های کوچک را می‌توان با جستجوی مستقیم و مقایسه مقدار داده شده با تک تک اعضاء مجموعه حل کرد. در این صورت، رویکرد مذکور به تعداد مقادیر مجموعه بستگی و بر روی داده‌های مرتب شده در حالت میانگین به $O(log\ n)$ مقایسه نیاز دارد. اما هنگامی که بر روی مجموعه‌های عظیم از مقادیر در برنامه‌های داده بزرگ اجرا شوند چنین رویکردی کارآمد نیست، و به زمان و حافظه O(n) برای ذخیره مقادیر نیاز دارد. از راه‌حل‌های ممکن تقسیم چنین مجموعه‌هایی به قطعات و اجرای مقایسه‌ها به صورت موازی است که به کاهش زمان محاسبه کمک می‌کند. اما، به دلیل پردازش داده بزرگ لزوما ذخیره چنین مجموعه‌های عظیمی از مقادیر عملی تقریباً ناممکن است و همیشه قابل اجرا نیست. از سوی دیگر، در بسیاری از موارد، لازم نیست دقیقاً دانست کدام عضوِ مجموعه مطابقت داده شده است، بلکه صرفا کفایت می‌کند که وجود تطابق تایید شود. در چنین مواردی می‌توان از امضا یا اثرانگشت مقادیر به جای مقدار کامل آنها استفاده کرد و صرفا امضا را ذخیره کرد.

مثال ۱- مسئله مرور ایمن

در طراحی مرورگر وب بهتر است از ورود کاربر به URL-های حاوی بدافزار و دسترسی او به آنها جلوگیری کرد. پس، در صورت درخواست کاربر برای بازدید از آن صفحات، مرورگر یا باید به کاربر هشدار دهد یا حتی از بازدید وی جلوگیری کند. راه‌حل فوری که ترافیک شبکه را به حداقل می‌رساند، ذخیره تمامی چنین URL-هائی در برنامه است. هنگامی که کاربر نشانی را وارد می‌کند بدافزاری آن بررسی می‌شود و در صورت عدم بدافزاری به کاربر اجازه دسترسی می‌دهد. چنین پیاده‌سازی ساده‌ای تا زمانی که تعداد URL-های مخرب کم باشد روش و گزینه‌ای مناسب است. اما، در دنیای واقعی و افزایش تعداد نشانی‌ها چنین راه‌حلی چندان کارآمد نیست. در عوض، به ساختاری ویژه نیاز خواهیم داشت که می‌تواند URL-های مخرب، یا صرفا بخشی از اطلاعات درباره آنها، را بدون رشد خطی در اندازه زمانی که URL-ی جدید معرفی می‌شود، ذخیره کند. همچنین، از دیگر الزامات پشتیبانی از بررسی فهرست شدن URL و تسریع تا حد امکان است تا کاربران مدتی طولانی را در انتظار نمانند.

مسئله عضویت علاوه بر اهمیت و کاربرد در علم رایانه، در ابزارها و تحقیقات دیگر شاخه‌هانیز اهمیتی به سزا ایفا می‌کند.

مثال ۲- توالی‌های دی‌ان‌ِای

از مسائل مهم در مطالعات متاژنومیک، طبقه‌بندی توالی‌ها به عنوان «جدید» یا متعلق به ژنوم شناخته شده، و فیلتر و نادیده‌گیری داده‌هایی که قبلاً مشاهده شده است. معمولا در مرحله پیش‌پردازش موارد مربوط به عضویت اجرا می‌شود و اگر به طور کارآمد انجام شود، می‌تواند پیچیدگی داده‌ها را قبل از انجام تحلیل دقیق‌تر کاهش دهد. مسئله جستجوی سریع را می‌توان با استفاده از مقدار درهم حل کرد. به بیان دیگر، کلید هر عضو از مجموعه داده با تابعی درهم‌ساز ایجاد و با مقادیر در جدول مقایسه می‌شود. با وجود اینکه، چنین رویکردی احتمال کمی از خطا (ناشی از تصادم‌های احتمالی درهم‌ساز) را ایجاد می‌کند، باز به حدود O(log n) بیت برای هر مقدار درهم نیاز دارد که برای مجموعه‌های داده عظیم در عمل غیرقابل اجرا است.

در این فصل، راه‌حل‌های متفاوت از جدول‌های درهم معمولی را در نظر می‌گیریم که به فضای کمتری نیاز دارند، جستجوهای سریع‌تری انجام می‌دهند، و احتمالات خطای کمتری را تضمین می‌کنند. چنین داده‌ساختارهای کارآمد به کاهش فضای لازم برای مدیریت حجم بالای داده کمک می‌کنند و امکان اجرای پرسش‌های عضویت با عملکرد خوب را فراهم می‌آورند. ابتدا فیلتر بلوم را معرفی می‌کنیم. سپس، بهبودها و نسخه‌های بعدی آن تشریح می‌شود و در نهایت، جانشین‌های متاخر آن را بیان می‌کنیم.

فیلتر بلوم

فیلترهای بلوم به سازوکاری استانده در بررسی مسئلهٔ عضویت در سامانه‌هایی بدل شده‌اند که با مجموعه‌داده‌های بسیار بزرگ سروکار دارند. کاربرد گستردهٔ آن‌ها، به‌ویژه در شبکه‌ها و پایگاه داده‌های توزیع‌شده، از کارآمدی قابل‌توجه آنها در شرایطی ناشی می‌شود که به نوعی کارکرد مشابه جدول درهم نیاز است، اما فضای کافی در اختیار نیست. این داده‌ساختار را بارتون بلوم در ۱۳۴۹ معرفی کرد. اما تا زمان حاضر توجه چندانی به آن نشد، و در پی افزایش نیاز به مهار و فشرده‌سازی مجموعه‌داده‌های عظیم طی چند دههٔ اخیر بود که «شکوفائی» این ساختار فراتر از «عنوان صرف» شد. افزون بر این، هنوز فیلترهای بلوم خاصه و مسئله عضویت عموما محل تحقیق در جامعهٔ پژوهشی علم رایانه هستند و در پی برطرف‌کردن برخی محدودیت‌های این ساختار و سازگاری آن با نیازمندی‌های متفاوت، گونه‌های متعددی بر مبنای نسخهٔ پایهٔ آن طراحی و توسعه داده شده است.

این داده‌ساختار احتمالاتی فضا-کارآمد برای نمایش مجموعه داده $D = \{x_1, x_2, \dots, x_n\}$ از n مقدار است که از دو عملیات درج (افزودن مقدار به مجموعه)، و آزمون عضویت مقداری در مجموعه پشتیبانی می‌کند. فیلتر بلوم می‌تواند مجموعه بزرگ را با اجتناب از ذخیرهٔ مقدار دقیق و عین به عین اعضا به صورتی سخت کارآمد ذخیره کند. این فیلتر صرفا مجموعه‌ای «تقریباً» منحصربه‌فرد از بیت‌ها را ذخیره می‌کند که مربوط به تعدادی تابع درهم‌ساز است که الگوریتم روی مقادیر ورودی اعمال می‌کند. پس، فیلتر بلوم با آرایه‌ای بیتی به طول m و تعداد k تابع درهم‌ساز مختلف $\{h_i\}_{i=1}^{k}$ نمایش داده می‌شود. فرض می‌شود که اندازهٔ m متناسب با تعداد مقادیر مورد انتظار n است، اما تعداد k بسیار کوچکتر از m است. توابع درهم‌ساز h_i باید مستقل و به طور یکنواخت توزیع شده باشند. به این ترتیب، مقادیر درهم را به طور یکنواخت تصادفی در فیلتر توزیع می‌شود (می‌توان توابع درهم‌ساز را چون نوعی مولد اعداد تصادفی در نظر آورد) و احتمال تصادم درهم را کاهش می‌یابد. چنین رویکردی فضای ذخیره‌سازی را سخت کاهش می‌دهد و بدون توجه به تعداد مقادیر در داده‌ساختار و اندازه آنها، با رزرو چند بیت برای هر مقدار، به تعداد ثابتی بیت نیاز دارد.

در ابتدا فیلتر خالی و در نتیجه همه بیت‌ها در آرایهٔ بیتی برابر با صفر است. برای درج مقدار x در فیلتر، تمامی توابع درهم‌ساز h_i را بر ورودی x اعمال و مقدارهای متناظر $j = h_i(x)$ را محاسبه و بیت متناظر j را در فیلتر برابر یک قرار می‌دهد. در خور ذکر است برخی از بیت‌های آرایه که قبلا مقدار یک گرفته‌اند ممکن است به دلیل تصادم‌های درهم چندین بار دیگر برای تنظیم انتخاب شوند. در چنین مواقعی با اولین انتخاب مقدار آن برابر یک قرار و در انتخاب‌های بعدی مقدار آن بدون تغییر و برابر یک باقی می‌ماند. الگوریتم ۱ نحوه درج در آرایه متناظر داده ساختار ببلوم را نمایش می‌دهد.

الگوریتم ۱- درج مقدار در فیلتر بلوم

ورودی: مقدار $x \in D$

ورودی: فیلتر بلوم با k تابع درهم‌ساز $\{h_i\}_{i=1}^{k}$

برای i از ۱ تا k انجام بده

$j \leftarrow h_i(x)$

$BloomFilter[j] \leftarrow 1$

با توجه به الگوریتم، مثال زیر چند درج در فیلتری را نشان می‌دهد.

مثال ۳- درج در فیلتر: فیلتر بلوم با طول $m = 10$ و دو تابع درهم‌ساز ۳۲ بیتی مورمور۳ و فن‌و-۱الف برای تولید مقادیر در محدوده $\{0,1, \dots, m-1\}$ را در نظر می‌گیریم:

$$h_1(x) = MurmurHash3(x)\%m$$
$$h_2(x) = FNV1a(x)\%m$$

فیلتر خالی به صورت زیر است.

۰	۱	۲	۳	۴	۵	۶	۷	۸	۹
0	0	0	0	0	0	0	0	0	0

فرضا، نام پایتخت‌ها را در فیلتر درج می‌کنیم. با کپنهاگ شروع کنیم و برای یافتن بیت‌های متناظر در فیلتر، مقادیر درهم آن را محاسبه می‌کنیم:

$$h_۱(\text{Copenhagen}) = MurmurHash3(\text{Copenhagen})\%۱۰ = ۷$$
$$h_۲(\text{Copenhagen}) = FNV1a(\text{Copenhagen})\%۱۰ = ۳$$

از این رو، باید بیت‌های ۳ و ۷ را در فیلتر برابر یک تنظیم کنیم:

۰	۱	۲	۳	۴	۵	۶	۷	۸	۹
0	0	0	۱	0	0	0	۱	0	0

ممکن است مقادیر مختلف بیت‌های متناظر مشترکی داشته باشند. مثلا، با افزودن دوبلین داریم:

$$h_۱(\text{Dublin}) = MurmurHash3(\text{Dublin})\%۱۰ = ۱$$
$$h_۲(\text{Dublin}) = FNV1a(\text{Dublin})\%۱۰ = ۳$$

همان‌طور که از مقادیر و جدول برمی‌آید، موقعیت‌های بیت متناظر آن در فیلتر مدخل‌های ۱ و ۳ هستند، که بیت مدخل ۱ قبلاً تنظیم نشده است ولی مدخل ۳ برابری مقدار دودوئی یک قرار گرفته است. این بدان معناست که بعضی از اعضای در فیلتر به جز دوبلین ، مدخل ۳ را به عنوان یکی از بیت‌های متناظر خود دارند:

۰	۱	۲	۳	۴	۵	۶	۷	۸	۹
0	۱	0	۱	0	0	0	۱	0	0

در صورت نیاز به بررسی وجود مقدار x در فیلتر، همهٔ k تابع درهم‌ساز $h_i = \{h_i(x)\}_{i=۱}^{k}$ را محاسبه کرده و بیت‌ها را در موقعیت‌های متناظر بررسی می‌کنیم. اگر همه بیت‌ها روی یک تنظیم شده باشند، ممکن است مقدار x در فیلتر وجود داشته باشد. در غیر این صورت، مقدار x قطعاً در فیلتر وجود ندارد. عدم قطعیت در مورد عضویت مقداری ناشی از امکان مقداردهی بیت‌های آرایه حاصل از مقادیری است که قبلا اضافه شده‌اند (همانند مثال ۳). یا ممکن است حاصل از تصادم‌های شدیدی است که ناشی ازتصادم تصادفی همه توابع درهم‌ساز با یکدیگر است. الگوریتم ۲ آزمون عضویت مقداری در فیلتر بلوم را تعریف می‌کند.

الگوریتم ۲- آزمون عضویت مقدار در فیلتر بلوم

ورودی: مقدار $x \in D$

ورودی: فیلتر بلوم با k تابع درهم $\{h_i(x)\}_{i=۱}^{k}$

خروجی: غلط در صورت عدم عضویت مقدار x و درست در صورت احتمال وجود آ»

برای i از ۱ تا k انجام بده

$j \leftarrow h_i(x)$

اگر $BloomFilter[j] \neq ۱$

برگرداندن غلط

برگرداندن درست

مثال زیر با استفاده از الگوریتم ۲ وجود مقداری در فیلتری را گزارش می‌کند.

مثال ۴- آزمون مقادیر در فیلتر: فیلتر بلوم از مثال قبل را با دو مقدار نمایه شده، کپنهاگ و دوبلین داریم:

۰	۱	۲	۳	۴	۵	۶	۷	۸	۹
0	۱	0	۱	0	0	0	۱	0	0

برای آزمون وجود مقدار کپنهاگ در فیلتر، در ابتدا مقادیر درهم‌ساز آن $h_۱(Copenhagen) = ۷$ و $h_۲(Copenhagen) = ۳$ را محاسبه می‌شود. پس از آن، بیت‌های متناظر را در فیلتر بررسی و برابر یک بودن هر دو روشن می‌شود، پس می‌توان ادعا کرد که کپنهاگ محتملا در فیلتر موجود است.

حال شهر رُم را در نظر می‌گیریم و درهم‌های آن را برای یافتن بیت‌های متناظر در فیلتر محاسبه می‌کنیم:

$$h_۱(\text{Rome}) = MurmurHash3(\text{Rome})\%۱۰ = ۵$$
$$h_۲(\text{Rome}) = FNV1a(\text{Rome})\%۱۰ = ۶$$

با بررسی بیت‌های ۵ و ۶، روشن می‌شود که بیت ۵ تنظیم نشده است، بنابراین مقدار رم قطعاً در فیلتر نیست و دیگر لزومی به بررسی بیت ۶ نیست. هرچند ممکن است فیلتر پاسخ مثبت کاذب برگرداند. مقادیر درهم متناظر شهر برلین به صورت زیر است:

$$h_۱(\text{Berlin}) = MurmurHash3(\text{Berlin})\%۱۰ = ۱$$
$$h_۲(\text{Berlin}) = FNV1a(\text{Berlin})\%۱۰ = ۷$$

بیت‌های متناظر ۱ و ۷، هر دو در فیلتر تنظیم شده‌اند، بنابراین نتیجه تابع آزمون این است که تائید احتمالی عضویت رم در فیلتر است. هر چند به دلیل ایجاد فیلتر در مثال ۳ می‌دانیم که مقدار مذکور به فیلتر اضافه نشده بود و در نتیجه نمونه‌ای از تصادم درهم است. شایان ذکر است در این مورد خاص بیت ۱ با $h_۱(\text{Dublin})$ و بیت ۷ با $h_۲(\text{Dublin})$ تنظیم شده است.

اگر هر تابع درهم‌ساز $\{h_i(x)\}_{i=۱}^{k}$ در زمان ثابت قابل محاسبه باشد، که برای توابع درهم‌ساز پرکاربرد صادق است، زمان درج مقداری جدید یا آزمون وجود مقداری برابر مقدار ثابت $O(۱)$ و مستقل از طول فیلتر m و تعداد اعضای ثبت شده در فیلتر است. عملکرد فیلتر بلوم به توابع درهم‌ساز انتخاب شده سخت وابسته است. تابع درهم‌ساز با یکنواختی خوب، میزان مثبت کاذب مشاهده شده را کاهش می‌دهد. از سوی دیگر، هرچه محاسبه هر تابع درهم‌ساز سریع‌تر باشد، زمان کل هر عملیات کمتر است، و بنابراین توصیه به اجتناب از توابع درهم‌ساز رمزنگاری می‌شود.

مثال‌هائی که در ادامه آمده است نحوهٔ استفاده کاربردی از فیلتر بلوم را نشان می‌دهد.

مثال ۵- اسپفورد در سال ۱۳۷۰ روشی برای جلوگیری از رمزهای عبور به خطر افتاده با بهره از فیلتر بلوم پیشنهاد داد[۱]. صفحه ثبت‌نام سرویس وبی را در نظر می‌گیریم. قصد داریم از انتخاب رمز عبورهای ضعیف و پرریسک کاربران در صفحه ثبت‌نام جلوگیری شود. لازم به توجه است، در وب تاریک می‌توان صدها میلیون رمز عبور هک شده را یافت که می‌توانند در «حملهٔ لغت‌نامه»، یا حملهٔ جستجو کامل که با امتحان همه مقادیر از فهرستی از پیش آماده، تلاش‌های تکراری برای شکست احراز هویت انجام می‌دهد، استفاده شوند. بنابراین، هر بار که کاربر رمز عبور جدیدی تایپ می‌کند، می‌خواهیم مطمئن شویم که در چنین فهرستی نیست. با این حال، همراه با عدم امنیت مربوط به ذخیره رمزهای عبور خام، نمی‌خواهیم مجموعه داده عظیم را حفظ کنیم که با هر رمز عبور تازه اضافه شده به صورت خطی رشد کند و جستجوها را کُند کنَد (مانند پایگاه‌های داده سنتی). بنابراین، استفاده از فیلتر بلوم کارآمد از نظر فضا ضروری است. رویداد مثبت کاذب، در این مورد، موقعیتی است که به اشتباه فکر کنیم رمز عبور وارد شده نامناسب است. در چنین موارد نادری، باید از کاربر بخواهیم رمز عبور دیگری تایپ کند که معمولاً ضرری ندارد.

مثال ۶- کاربرد در اپلیکیشن موبایل بیت‌کوین: شبکه‌های همتابه‌همتا برای تبادل داده از فیلترهای بلوم بهره می‌گیرند و یکی از نمونه‌های شناخته‌شدهٔ آن، سامانهٔ بیت‌کوین است. تضمین شفافیت بین کاربران از ویژگی‌های مهم بیت‌کوین است. امری که از طریق آگاه بودن هر گره از تراکنش‌های سایر گره‌ها محقق می‌شود. با این وجود، برای گره‌هایی که روی تلفن‌های هوشمند یا دستگاه‌های مشابه با حافظه و پهنای باند محدود اجرا می‌شوند، نگه‌داری نسخهٔ کامل تمام تراکنش‌ها بسیار ناکارآمد و پرهزینه است. از این رو، بیت‌کوین امکان «تأیید سادهٔ پرداخت» (Simplified Payment Verification - SPV) را فراهم می‌کند، که طی آن گره می‌تواند

[۱] OPUS

با اعلام فهرستی از تراکنش‌های موردعلاقهٔ خود، به‌عنوان «گره سبک» عمل کند. امری که در تقابل با «گره‌های کامل» است که کل داده‌های زنجیره را نگه‌داری می‌کنند.
گره‌های سبک، فیلتر بلومی را که از فهرست تراکنش‌های مورد توجهشان ساخته شده است محاسبه و برای گره‌های کامل ارسال می‌کنند. بدین‌ترتیب، گرهِ کامل پیش از ارسال اطلاعات مربوط به تراکنش برای گره سبک، ابتدا فیلتر بلوم آن گره را بررسی می‌کند تا ارتباط تراکنش مربوطه را به گره سبک بسنجد. اگر «مثبت کاذب» رخ دهد، گره سبک می‌تواند پس از دریافت پیام، آن را نادیده بگیرد. در حال حاضر، بیت‌کوین روش‌های دیگری برای فیلترکردن تراکنش‌ها ارائه کرده است که از نظر امنیت و حریم خصوصی، بهبود یافته‌اند.

مثال ۷- اشتراک حافظه کش: فان و بوردر و همکاران در سال ۱۳۷۹ روشی کارآمد جهت اشتراک معرفی کردند.
فهرستی از پروکسی‌های کش توزیع شده $(P_1, P_2, \dots, P_n)$ در شبکه‌ای که حافظه کش‌های خود را به اشتراک می‌گذارند را در نظر می‌آوریم. در صورتی که محتوای URL درخواستی روی پروکسی P_i ذخیره شده باشد، پروکسی آن را بدون اتصال با سرور راه دور بازمی‌گرداند. در غیر این صورت، محتوا بازیابی می‌شود، به صورت محلی ذخیره می‌شود و برای مشتری ارسال می‌شود.
با هدف کمینه‌سازی ترافیک شبکه و توزیع ذخیره‌سازی، می‌توان مسیریابی را در شبکه پروکسی تنظیم و سعی کنیم به سمت پروکسی‌ای هدایت کنیم که قبلاً محتوا را ذخیره کرده است. در غیر این صورت، با سرور راه دور تماس بگیریم. از آنجایی که درخواست‌های مشتری می‌توانند به هر پروکسی بیایند، مشکل اشتراک‌گذاری مسیریابی در هر پروکسی وجود دارد و زمانی که تغییر می‌کند، تبادل آن در شبکه یا ادغام، در صورت لزوم، وجود دارد.
فیلتر بلوم انتخابی طبیعی برای ذخیرهٔ چنین فهرست‌های مسیریابی و انجام پرسش‌های عضویت سریع است. همچنین به دلیل اندازهٔ کوچک خود به راحتی در شبکه قابل انتقال است.
رویداد مثبت کاذب در پروکسی‌های بدین نحو است که یکی از پروکسی‌ها مثلا P_i فرض کند که پروکسی دیگری P_j ممکن است محتوای URL درخواستی را داشته باشد، اما در واقع ندارد. P_i ترافیک را به P_j هدایت می‌کند و از آن می‌خواهد محتوا را برگرداند، بنابراین P_j باید با سرور راه دور تماس بگیرد. در نتیجه، مقداری ترافیک شبکه اضافی تولید می‌کند و کپی‌های محلی اضافی را برای چنین محتوایی ذخیره می‌کند، که کاملاً قابل قبول است.

تمرین- بهبودها و راه‌حل‌های بعدی در حل مسئله عضویت در وب‌سرویس‌ها بر اساس کار اسپفورد و بیت‌کوین، فان را گزارش و پیاده کنید.

پیاده‌سازی ساده

پیاده‌سازی فیلتر بلوم پایه نسبتاً سرراست است. در این بخش، پیاده‌سازی ساده‌ای ارائه می‌شود که از بستهٔ پایتونی برای مورمور با نام mmh۳ که در فصل قبل معرفی شد استفاده می‌کند. با تنظیم مقادیر اولیهٔ متفاوت، می‌توان به مجموعه‌ای از توابع درهم مختلف دست یابیم. این پیاده‌سازی همچنین، از کتابخانهٔ bitarray استفاده می‌کند که امکان رمزگذاری کم‌حجم و کارآمد فیلتر را فراهم می‌آورد و برای اجرای کد باید آن را نصب کرد:

```
import math
import mmh3
from bitarray import bitarray


class BloomFilter:
```

```
    def __init_(self, n, f):
        self.n = n
        self.f = f
        self.m = self.calculateM()
        self.k = self.calculateK()
        self.bit_array = bitarray(self.m)
        self.bit_array.setall(۰)
        self.printParameters()

    def calculateM(self):
        return int(-math.log(self.f)*self.n/(math.log(۲)**۲))

    def calculateK(self):
        return int(self.m*math.log(۲)/self.n)

    def printParameters(self):
        print("Init parameters:")
        print(f"n = {self.n}, f = {self.f}, m = {self.m}, k = {self.k}")

    def insert(self, item):
        for i in range(self.k):
            index = mmh۳.hash(item, i) % self.m
            self.bit_array[index] = ۱

    def lookup(self, item):
        for i in range(self.k):
            index = mmh۳.hash(item, i) % self.m
            if self.bit_array[index] == ۰:
                return False
        return True
```

پیاده‌سازی مذکور را با افزودن چند رشتهٔ ساده امتحان می‌کنیم:

```
bf = BloomFilter(۱۰, ۰٫۰۱)
bf.insert("۱")
bf.insert("۲")
bf.insert("٤۲")

print("۱ {}".format(bf.lookup("۱")))
print("۲ {}".format(bf.lookup("۲")))
print("۳ {}".format(bf.lookup("۳")))
print("٤۲ {}".format(bf.lookup("٤۲")))
print("۴۳ {}".format(bf.lookup("٤۳")))
```

در سازندهٔ این پیاده‌سازی، کاربر می‌تواند حداکثر تعداد مقادیر (n) و میزان مطلوب مثبت کاذب (f) را مشخص کند، در حالی‌که خودِ سازنده دو پارامتر دیگر، یعنی m و k، را محاسبه و مقداردهی می‌کند. معمولاً اندازهٔ تقریبی مجموعه‌داده‌ای را که با آن سروکار داریم از پیش می‌دانیم و همچنین می‌توان سطح مقبول از احتمال مثبت کاذب را از قبل تبیین کرد. در نتیجه، رویکرد حاضر در

شبه کد معمول و رایج است. بخش‌های بعدی موارد لازم برای درک نحوهٔ تعیین این پارامترهای باقیمانده در پیاده‌سازی فوق، و نیز چگونگی پیکربندی فیلتر بلوم به‌گونه‌ای که بیشترین بازده را فراهم آورد توضیح می‌دهد.

ویژگی‌ها

الف- مثبت کاذب امکان‌پذیر است. همان‌طور که قبلاً ذکر شد، فیلتر بلوم به جای ذخیره مستقیم مقدار اصلی ورودی، صرفا به درهم‌های محاسبه شده متکی است که همگی در آرایه‌ای بیتی ذخیره می‌شوند. چنین نمایش کارآمدی از نظر فضا می‌تواند به موقعیت‌هایی منجر شود که برخی از مقادیر که عضو فیلتر نیستند (به فیلتر اضافه نشده‌اند)، عضویت آنها در آزمون عضویت الگوریتم ۲ به غلط تائید شود. چنین رویدادی مثبت کاذب نامیده می‌شود و به دلیل تصادم‌های درهم در بیت‌های ذخیره شده رخ می‌دهد. در عملیات آزمون هیچ دانش قبلی جهت تعیین ارتباط تنظیم بیتی خاص به مقداری که مقایسه می‌کنیم وجود ندارد و چنین رخدادهای مثبت کاذبی اجتناب‌ناپذیر است.

اصل فیلتر بلوم [بوردر ۸۳][۲]: هنگام استفاده از فهرست یا مجموعه، و با ارزشی فضا، در صورت امکان تا زمانی که اثر مثبت کاذب چندان بالا نرفته یا آستانه‌ای را رد نکرده است از فیلتر بلوم بهره گرفته شود.

موقعیت‌های مثبت کاذب به ندرت رخ می‌دهند و می‌توان احتمال آنها pr_{fp} را تخمین زد. یکی از روش‌های تخمین آن، محاسبهٔ مقدار کران پائینی است:

$$\mathrm{pr}_{fp} \approx \left(۱ - e^{-\frac{kn}{m}}\right)^k \qquad (۱)$$

مقدار احتمال مثبت کاذب به صورت زیر حاصل شده است. با فرض استقلال توابع درهم‌ساز از یکدیگر (نتیجهٔ تابعِ درهم بر نتایج سایر توابع تأثیری ندارد) و هر تابعِ درهم، کلیدها را به‌صورت یکنواخت تصادفی در بازهٔ $[۰, \ldots, m-۱]$ نگاشت می‌کند.
اگر t کسری از بیت‌های فیلتر بلوم باشد که پس از انجام n درج هنوز برابر ۰ هستند، و k تعداد توابع درهم‌سازی باشد، آنگاه احتمال رخداد یک مثبتِ کاذب (f) برابر با احتمال مشاهدهٔ k بیتِ یک است. زیرا تائید عضویت مقداری نیاز به وجود k بیت با مقدار ۱ دارد. دستیابی به k مقدار ۱ می‌تواند نتیجهٔ جست‌وجوی موفق برای عضوی باشد که واقعاً درج شده است. اما اگر پرس‌وجوها را به‌صورت یکنواخت تصادفی از دامنه‌ای بسیار بزرگ‌تر از مجموعه‌داده انتخاب کنیم، آنگاه احتمال مثبتِ واقعی، کسری ناچیز از این مقدار خواهد بود. مقدار t پیش از آن‌که تمام عملیات درج انجام شود ناشناخته است، زیرا به خروجی درهم بستگی دارد. اما به جای آن می‌توان از احتمالِ p بهره برد؛ احتمال این‌که یک بیت مفروض پس از انجام n درج همچنان برابر ۰ باقی بماند (p احتمال آن‌که بیتی معین پس از n درج مقداری برابر صفر است). این p از دید احتمالاتی همان درصد بیت‌های صفر موجود در فیلتر (همان t) را تقریب می‌زند. می‌توان نشان داد که مقدار p برابر است با:

$$pr = \left(۱ - \frac{۱}{m}\right)^{nk}$$

برای درک چرایی این موضوع، از فیلتر بلوم خالی شروع می‌کنیم. درست پس از آن‌که نخستین تابع درهم‌سازی $h_۱$ یکی از بیت‌ها را برابر ۱ قرار می‌دهد، احتمال این‌که بیت مفروض در فیلتر بلوم مقدار ۱ داشته باشد برابر $۱/m$ و احتمال صفر بودن آن برابر $۱ - \frac{۱}{m}$ است.

[۲] بوردر و همکاران تحلیل‌های بیشتری در مقاله مطالعه‌ای خود در سال ۱۳۸۳ مطرح کردند. خواننده علاقمند بدان مراجعه کند.

پس از آن‌که همهٔ k تابعِ درهم در درج نخست بیت‌های مربوطه را ۱ کردند، احتمال این‌که آن بیتِ ثابت همچنان ۰ باشد برابر $\left(۱ - \frac{۱}{m}\right)^{k}$ است. پس از درج n مقدار، احتمال برابر $\left(۱ - \frac{۱}{m}\right)^{nk}$ خواهد بود. در نتیجه، با استفاده از تقریب $\left(۱ + \frac{۱}{x}\right)^{x} \approx e$، داریم:

$$pr \approx e^{-\frac{nk}{m}}$$

با جاگذاری معادله (۱) حاصل می‌شود. اما صرفا اکتفا به این مقدار شاید این شائبه را ایجاد کند که درصد واقعی صفرها ممکن است دارای انحراف چشمگیری از مقدار حاصل باشد. کران‌ها خاصه کران چرنوف در اینجا جهت اطمینان از مقدار حاصل کمک کننده است.

کران چرنوف قضیه‌ای است که احتمال انحراف شدید متغیر تصادفی از میانگینش را محدود می‌کند. با استفاده از آن می‌توان نشان داد که کسری از صفرها در فیلتر بلوم به‌شدت حول مقدار میانگین خود متمرکز است. بیان عمومی کران چرنوف برای متغیرهای تصادفی X که جمعی از متغیرهای دنباله‌ایِ مستقلِ دوتایی‌اند صدق می‌کند. متغیر X را به‌صورت تعداد کل بیت‌های صفر موجود در فیلتر بلوم جایی که $X_i = ۰$ اگر بیت i–ام دارای مقدار ۱ و $X_i = ۱$ در غیر این‌صورت باشد تعریف می‌کنیم.

با استفاده از کران چرنوف نشان می‌دهیم که مقدار X انحراف قابل‌توجهی از میانگینش ندارد. در مسئلهٔ مذکور، متغیرهای X_i کاملاً مستقل نیستند، اما اندکی هم‌بستگی منفی دارند (که حتی مطلوب‌تر است!). اگر مقدار بیتی برابر مقدار ۱ شود احتمال ۱ شدن سایر بیت‌ها را اندکی کاهش می‌دهد. بیان کلی کران بالای چرنوف (و می‌توان مورد مشابهی برای کران پایین نیز به‌کار برد)، با μ به‌عنوان میانگین متغیر تصادفی X، به‌صورت زیر است:

$$Pr[X \geq (۱ + \delta)\,\mu] \leq e^{-\frac{\mu\delta^{۲}}{۳}}$$

وقتی آن را برای مسئلهٔ خود به‌کار ببریم، $\mu = E[X] = mp = me^{-\frac{nk}{m}}$ خواهد بود. اگر $\delta = ۱$ بگیریم و مقدار آن را در کران چرنوف قرار دهیم، احتمال این‌که X بیش از دو برابر مقدار میانگین خود انحراف پیدا کند برابر خواهد شد با:

$$pr[X \geq ۲\,\mu] \leq e^{-\frac{\mu}{۳}}$$

می‌توان با اطمینان فرض کرد که $nk = \theta(m)$. این بدین معناست که احتمال انحراف X از میانگین به‌اندازهٔ بیش از یک ضریب ۲، به‌صورت نمایی کوچک است (به‌صورت $e^{-\theta(m)}$).

بنابراین، p تقریبِ بسیار خوبی برای t است. یعنی درصد صفرهای فیلتر بلوم. این امر توجیه می‌کند که چرا در معادله (۲) می‌توان t را با p جانشین کرد. به این ترتیب، استخراج و اثبات معادله تکمیل می‌شود.

همان‌طور که در کران بالا قابل مشاهده است، با تعداد ثابت اعضاء مورد انتظار n، احتمال مثبت کاذب به انتخاب k و m بستگی دارد و نیاز به بررسی و سبک-سنگینی بین طول فیلتر، تعداد توابع درهم‌ساز، و احتمال چنین رویدادهایی را نشان می‌دهد.

در بدترین حالت مربوط به یک شدن همهٔ بیت‌ها یا پر بودن فیلتر است، هر جستجو پاسخ مثبت (کاذب) بر می‌گرداند. این بدان معناست که انتخاب m به تعداد (تخمینی) مقادیر n که انتظار می‌رود اضافه شوند بستگی دارد، و m باید نسبت به n بسیار بزرگ باشد.

در عمل، طول فیلتر m، با توجه به احتمال مثبت کاذب pr_{fp} و تعداد مقادیر مورد انتظار n، می‌تواند با معادله زیر تبیین شود:

$$m = -\frac{n \ln \mathrm{pr}_{fp}}{(\ln ۲)^{۲}} \qquad (۲)$$

بنابراین، در صورت نیاز به حفظ احتمال مثبت کاذب نیاز است که فیلتر با نسبتی خطی با تعداد مقادیر رشد کند. با داشتن نسبت $\frac{m}{n}$، به معنای تعداد بیت‌های تخصیص داده شده به ازای هر عنصر، احتمال مثبت کاذب را می‌توان با انتخاب تعداد توابع درهم‌ساز k تنظیم کرد. انتخاب بهینه k با کمینه کردن احتمال مثبت کاذب در معادلهٔ (۱) محاسبه می‌شود:

$$k = \frac{m}{n} \ln ۲ \qquad (۳)$$

به عبارت دیگر، تعداد بهینه توابع درهم‌ساز k حدود ۰.۷ تعداد بیت‌ها به ازای هر مقدار است. جدول زیر برخی مقادیر k برای راه‌حل‌های نزدیک به بهینه را نشان می‌دهد.

pr_{fp}	$\frac{m}{n}$	K
۰.۰۵۶۱	۶	۴
۰.۰۲۱۵	۸	۶
۰.۰۰۳۱۴	۱۲	۸
۰.۰۰۰۴۵۸	۱۶	۱۱

جدول ۱ انتخاب‌های نسبتا بهینهٔ پارامترها

مثال ۸- تخمین پارامترها: با توجه به معادلهٔ تخمین اندازه فیلتر، برای داشتن احتمال مثبت کاذب $\mathrm{pr}_{fp} = ۱\%$، نیاز است طول فیلتر ده برابر تعداد مقادیر مورد انتظار n باشد و از شش تابع درهم‌ساز استفاده کند. در خور ذکر است طول فیلتر به اندازه خود مقادیر ورودی بستگی ندارد و برای مقادیر با ماهیت‌های مختلف ثابت می‌ماند.

مثال ۹- حافظه مورد نیاز: با توجه به معادله تخمین m، جهت مدیریت یک میلیارد مقدار و حفظ احتمال رویدادهای مثبت کاذب حدود دو درصد و استفاده از تعداد بهینه‌ای از توابع درهم‌ساز، نیاز به انتخاب فیلتری به طول $m = -۱۰^{۹} \times \frac{\ln ۰.۰۲}{(ln ۲)^{۲}} \approx ۸.۱۴ \times ۱۰^{۹}$ بیت انتخاب کنیم که تقریباً برابر با یک گیگابایت حافظه است.

فیلترهای بلوم را می‌توان چون تعمیمی از جدول‌های درهم در نظر گرفت. در واقع، فیلتری با یک تابع درهم‌ساز معادل جدول درهم است. ولی، با استفاده از توابع درهم‌ساز متعدد، فیلترهای بلوم می‌توانند احتمال مثبت کاذب ثابت را برای تعداد ثابتی از بیت‌ها به ازای هر مقدار حفظ کنند، در حالی که جدول‌های درهم چنین قابلیتی را پشتیبانی نمی‌کنند.

ب- منفی کاذب امکان‌پذیر نیست. در مقابل وضعیت فوق، اگر خروجی فیلتر بلوم عدم عضویت عنصری باشد، مقدار مذکور قطعاً در مجموعه نیست:

$$\mathrm{pr}_{fn} = ۰$$

ج- فیلتر بلوم تا زمانی که در حافظه جا شود به خوبی کار می‌کند. همانطور که در بالا ذکر شد، احتمال مثبت کاذب را می‌توان با تخصیص حافظه بیشتر کاهش داد، به همین دلیل است که افراد تمایل به ایجاد فیلترهای بزرگتر (با m بزرگتر) دارند. با این حال، چنین فیلترهای بلوم کلاسیکی تا هنگامی که حافظه گنجایش آنها را داشته باشد به خوبی کار می‌کنند. به محض اینکه بیش از حد بزرگ شد، لاجرم و به اجبار انتقال به دیسک شوند. چنین انتقالی با مشکل ناشی از طراحی مواجه می‌شود. به بیان دیگر، توابع درهم‌ساز با توزیع یکنواخت، شاخص‌های متناظر تصادفی تولید می‌کنند که هر بار نیاز به دسترسی تصادفی دارند. نیاز مذکور برای

دیسک‌های با صفحه‌های گردان و نوک‌های خواندن-نوشتن متحرک بسیار ناکارآمد است. با وجود سرعت بهتر دستگاه‌های ذخیره‌سازی حالت جامد، اما هنوز هم در حد کفایت نیست و در نتیجه استفاده از الگوریتم بلوم ناکارآمد باقی می‌ماند.

دو فیلتر بلوم متفاوت با طول یکسان را تنها در صورتی می‌توان ادغام کرد که توابع درهم‌ساز یکسانی نیز داشته باشند. در این حالت، ادغامْ به عملیات یای بیتی تحویل می‌شود و حاصل آن با فیلتر بلوم حاصل از اتحاد آن دو مجموعه از مقادیر کاملاً متناظر است. اشتراک دو فیلتر بلوم نیز ممکن است و با وصل (and) بیتی صورت می‌پذیرد (چرا؟)، با این حال، نتیجه می‌تواند احتمال تصادم بیشتری داشته باشد.
در صورت اتمام فضای فیلتر بلوم و نیاز به افزایش فضای آن، این‌کار بدون محاسبه مجدد تمام درهم‌هائی که قبلا در فیلتر قرار گرفتند ممکن نیست. ناگفته پیداست چنین کاری در کاربردهای کلان‌داده‌ای ناممکن است.

د- حذف امکان‌پذیر نیست. برای حذف عنصری خاص از فیلتر بلوم باید بیت‌های k متناظر آن را در آرایه بیتی پاک کرد. متأسفانه، یک بیت واحد می‌تواند به دلیل تصادم‌های درهم‌ساز و بیت‌های مشترک بین عناصر، با چندین مقدار مطابقت داشته باشد.
چند بهبود جهت پشتیبانی حذف معرفی شده‌اند، اما معمولا یک پای ماجرا می‌لنگد و هزینه‌هایی از نظر فضا و سرعت تحمیل می‌کند و نیاز به تحقیقات بیشتر کماکان وجود دارد. سخن کوتاه، به دلیل سرعت و کارآمدی، فیلتر بلوم کلاسیک استفاده فراوانی دارد. نکته مثبت این است که عدم پشتیبانی از حذف در بسیاری از برنامه‌های کاربردی دنیای واقعی مشکلی ایجاد نمی‌کند، اما اگر واقعاً به آن نیاز باشد، چاره‌ای جز به بهبود فیلتر بلوم مانند «فیلتر بلوم شمارنده» نیست.

شمارش مقادیر منحصربه‌فرد در فیلتر

ساملیاس و پیر بالدی روشی برای تخمین تعداد مقادیر منحصربه‌فرد اندیس شده در فیلتر را معرفی کردند که، در واقع، بهبودی بر «الگوریتم شمارش خطی» است که در بخش‌های بعد بحث می‌شود. با استفاده از اطلاعات مربوط به تعداد بیت‌های تنظیم شده در فیلتر و احتمال تنظیم هر بیت، معادله ساده‌ای برای تقریب تعداد اعضاء موجود در فیلتر عرضه می‌کند. از آنجایی که دو مقدار یکسان که به فیلتر اضافه می‌شوند، تعداد بیت‌های تنظیم شده را تغییر نمی‌دهند، چنین تقریبی تخمینی برای تعداد مقادیر منحصربه‌فرد (که به عنوان تعداد اصلی شناخته می‌شود) ارائه می‌دهد. الگوریتم زیر رویهٔ تقریب را نشان می‌دهد

الگوریتم ۳- شمارش مقادیر منحصربه‌فرد در فیلتر بلوم
ورودی: فیلتر بلوم به طول m با k تابع درهم‌ساز
خروجی: تعداد مقادیر منحصربه‌فرد در فیلتر
N برابر شمارش Bloomfilter[j] برای $j = ۱, \dots, m$
اگر $N < k$ آن‌گاه برگرداندن صفر
اگر N==k آن‌گاه برگرداندن یک
اگر N==m آن‌گاه برگرداندن m/k
برگرداندن $-\frac{m}{k}.\ln\left(۱-\frac{N}{m}\right)$

احتمال صفر بودن بیتی پس از درج n عضو جدید برابر است با

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

در نتیجه، میانگین نسبت یک‌ها برابر است با

$$\frac{E[N]}{m} = ۱ - e^{-kn/m}$$

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

$$e^{-kn/m} = ۱ - \frac{N}{m}$$

$$-\frac{kn}{m} = \ln\left(۱ - \frac{N}{m}\right)$$

$$n = -\frac{m}{k}\ln\left(۱ - \frac{N}{m}\right) \qquad (۴)$$

که دقیقا برابر مقداری است که الگوریتم وقتی $۰ < N < m$ برمی‌گرداند. حالت‌های خاص آن عبارت است از الف- $N < k$ که غیرممکن است یا چند مورد از درهم‌سازها بیت‌های یکسانی را برای ورودی خاصی تنظیم کنند مگر $n = ۰$. ب- $N = k$ که نشان می‌دهد احتمالا یک عضو افزوده شدن. ج- $N = m$ یا تمامی بیت‌ها برابر ۱ است. در نتیجه ظرفیت بیشینه یا m/k عضو دارد. معادلهٔ $-\frac{m}{k}\ln(۱ - N/m)$ تقریبی مناسب از n در مواقعی است که فیلتر اشباع نشده است.

فیلتر بلوم شمارنده

فیلتر بلوم شمارنده از اصلاحات مهم بر فیلتر بلوم کلاسیک است که از حذف پشتیبانی می‌کند که لی فن، پی کائو، جوسارا آلمیدا، و آندری برودر در سال ۱۳۷۹ معرفی کردند [فان ۷۹]. با تکیه بر الگوریتم فیلتر بلوم کلاسیک، آرایه‌ای از m شمارنده $\{C_j\}_{j=۱}^{m}$ متناظر با هر بیت در آرایه فیلتر را معرفی می‌کند. «فیلتر بلوم شمارنده» با افزوده شدن هر مقدار مدخل‌های متناظر را یک واحد افزایش می‌دهد و در نتیجه تقریبی از تعداد دفعاتی مشاهدهٔ هر مقدار را در فیلتر فراهم می‌کند. داده‌ساختار مرتبط شامل آرایه‌ای بیتی و آرایه‌ای از شمارنده‌ها هر دو به طول m است که همگی به صفر مقداردهی اولیه می‌شوند.

هنگامی مقداری جدید در فیلتر بلوم شمارنده درج می‌شود، ابتدا موقعیت‌های بیت متناظر آن را محاسبه می‌شود، و سپس برای هر موقعیت، شمارنده متناظر افزایش می‌یابد. لازم به ذکر است در آرایهٔ بیتی زمانی بیت از صفر به یک تغییر می‌کند که شمارنده از صفر به یک تغییر می‌کند. به بیان دیگر، شبیه الگوریتم بلوم کلاسیک عمل می‌کند.

الگوریتم ۴- درج در فیلتر بلوم شمارنده
ورودی: مقدار $x \in D$
ورودی: فیلتر بلوم شمارنده با m شمارنده $\{C_j\}_{j=۱}^{m}$ و k تابع درهم‌ساز $\{h_i\}_{i=۱}^{k}$
برای i از ۱ تا k انجام بده
$j \leftarrow h_i(x)$
$C_j \leftarrow C_j + ۱$
اگر $C_j == ۱$
$CountingBloomFilter[j] \leftarrow ۱$

عملیات آزمون دقیقاً همانند فیلتر بلوم معمول عمل می‌کند، زیرا اصلاً نیازی به بررسی شمارنده‌ها نیست. به دلیل آرایه‌های بیتی یکسان فیلترها، مقدار زمان مورد نیاز برای آزمون عنصری با الگوریتم کلاسیک یکسان است.

الگوریتم ۵- آزمون عضویت مقدار در فیلتر بلوم شمارنده

ورودی: مقدار $x \in D$

ورودی: فیلتر بلوم شمارنده با k تابع درهم‌ساز $\{h_i\}_{i=۱}^{k}$

خروجی: غلط درصورت نیافتن مقدار و درست در صورت وجود عنصر

برای i از ۱ تا k انجام بده

$j \leftarrow h_i(x)$

اگر $CountingBloomFilter[j] \neq ۱$

برگرداندن غلط

برگرداندن درست

حذف برعکس درج عمل می‌کند. برای حذف مقدار x، هر k مقدار درهم را با توابع $\{h_i\}_{i=۱}^{k}$ محاسبه کرده و شمارنده‌های متناظر را کاهش می‌دهیم. اگر شمارنده مقدار خود را از یک به صفر تغییر دهد، بیت متناظر در آرایه بیتی باید پاک شود. الگوریتم زیر فرایند حذف عضوی را از فیلتر بلوم شمارنده نشان می‌دهد.

الگوریتم ۶- حذف مقدار از فیلتر بلوم شمارنده

ورودی: مقدار $x \in D$

ورودی: فیلتر بلوم شمارنده با m شمارنده $\{C_j\}_{j=۱}^{m}$ و k تابع درهم‌ساز $\{h_i\}_{i=۱}^{k}$

برای i از ۱ تا k انجام بده

$j \leftarrow h_i(x)$

$C_j \leftarrow C_j - ۱$

اگر $C_j == ۰$

$CountingBloomFilter[j] \leftarrow ۰$

الگوریتم حذف مقدار فرض می‌کند که مقدار x در فیلتر وجود دارد (یا ممکن است وجود داشته باشد)، که در عمل لزوما این‌گونه نیست. بنابراین، بهتر است قبل از اعمال الگوریتم کاهش شمارنده‌های متناظر، الگوریتم آزمون مقدار اجرا شود.

فیلتر بلوم شمارنده تمام ویژگی‌های فیلتر بلوم معمول را از جمله تخمین خطای مثبت کاذب و معادلات انتخاب بهینه m و k را به ارث برده است. طبعا، فیلترهای بلوم شمارنده بسیار بزرگتر از فیلترهای بلوم کلاسیک هستند زیرا به حافظه اضافی برای شمارنده‌ها ولو اینکه مقدار بیشتر مدخل‌ها صفر باشند نیاز دارند. در نتیجه، تخمین میزان بزرگ شدن چنین شمارنده‌هائی و چگونگی وابستگی اندازهٔ آنها به طول فیلتر m و تعداد توابع درهم‌ساز k اهمیتی به سزا دارد. با فرض اینکه هر شمارنده C دارای سطح ظرفیت N است، احتمال فراتر رفتن مقدار از N (به معنای سرریزی) در فیلتر بلوم شمارنده با طول m با انتخاب بهینه k برابر مقدار زیر است:

$$pr(\max(C) \geq N) \leq m \times \left(\frac{eln2}{N}\right)^{N} \qquad (۵)$$

تمرین- تحقیق کنید.

مثال ۱۰- شمارنده‌های ۴ بیتی $N = ۱۶$، احتمال سرریز با معادله بالا برابر است با

$$pr(\max(C) \geq ۱۶) \leq m \times ۱.۳۷ \times ۱۰^{-۱۵}$$

به عبارت دیگر، اگر ۴ بیت به ازای هر شمارنده تخصیص دهیم، احتمال سرریز برای مقادیر عملی m (به عنوان مثال، چند میلیارد موقعیت بیتی) در طول درج اولیه در فیلتر بسیار کم است. پس از حذف و درج‌های متعدد، احتمال می‌تواند کمی بیشتر شود، اما هنوز برای استفاده عملی به اندازه کافی کم است.

برای جلوگیری از سرریز حسابی (یعنی افزایش شمارنده‌ای که قبلاً حداکثر مقدار ممکن را دارد)، هر شمارنده در آرایه باید به اندازه کافی بزرگ باشد تا ویژگی‌های فیلترهای بلوم حفظ شود. در عمل، شمارنده از ۴ یا چند بیت تشکیل شده است و فیلتر بلوم شمارنده، بنابراین، چهار برابر فضای بیشتری نسبت به فیلتر بلوم معمول نیاز دارد. اگر شمارنده‌ای چهار بیتی از مقدار ۱۵ فراتر رفت، می‌توانیم به سادگی آن را «به حد بالا» تحویل و صرفا مقدار در ۱۵ بماند. هر چند ممکن است پس از حذف‌های زیاد منجر به تولید پاسخ منفی در فیلتر بلوم شمارنده شود (شمارنده زمانی که نباید صفر شود صفر می‌شود)، هرچند احتمال چنین زنجیره‌ای از رویدادها بسیار کم و ناچیز است.

تمرین- میزان احتمال را بسنجید.

با این حال، می‌توان نسخه پیچیده‌تری از فیلتر بلوم شمارنده با شمارنده‌های کوچکتر (مثلا دو بیتی) طراحی کرد که فضای کمتری اشغال می‌کند و در عوض با اتخاذ رویکردی مشابه آدرس‌دهی بسته در جدول‌های درهم‌ساز و معرفی جدول درهم‌ساز ثانویه برای مدیریت شمارنده‌های سرریز مسئله را حل کرد.

تمرین- تحقیق و کارآمدی آن را تحلیل کنید.

بنابراین، فیلتر بلوم شمارنده به دلیل احتمال مثبت خطا به دلیل سرریز شدن شمارنده صرفا از حذف درست به صورت احتمالی پشتیبانی می‌کند. با وجود ایرادهای مورد اشاره، فیلترهای بلوم شمارنده در Apache Hadoop و Apache Spark در برنامه MapReduce برای تسریع پردازش مجموعه‌های داده عظیم روی خوشه‌های بزرگ با کمک به کاهش حجم سخت مورد استفاده‌اند.

تمرین- فیلتر بلوم شمارنده چه مسئله‌ای را در برنامه‌های مذکور حل می‌کند؟

فیلتر خارج‌قسمت

فیلتر خارج قسمت را مایکل بندر و همکاران در سال ۱۳۹۰ پیشنهاد دادند که از داده‌ساختارهایی است که علاوه بر پشتیبانی از عملیات اصلی فیلترهای بلوم، محلی‌سازی داده را بهبود می‌دهد و نیاز به تعداد کمی از دسترسی‌های دیسک پیوسته را ممکن می‌سازد. فیلتر مذکور نه تنها دارای عملکرد فضا و زمان مشابه است، بلکه از حذف پشتیبانی و امکان تغییر اندازه یا ادغام پویا را فراهم می‌سازد. نام این داده‌ساختار از خارج‌قسمت نتیجه عملیات تقسیم ریشه می‌یابد.

فیلتر خارج‌قسمت در ساده‌ترین شکل خود جدول درهم است که از کاوش خطی استفاده می‌کند. اما به جای ذخیره‌سازی مقدارهای کلید در خانه‌ها، مانند جدول درهم خطی کلاسیک، فیلتر خارج‌قسمت مقدارهای درهم (اصطلاح اثرانگشت به جای درهم نیز به کار خواهد رفت) را ذخیره می‌کند. به عبارت دقیق‌تر، فیلتر خارج‌قسمت تکه‌ای از هر درهم را ذخیره می‌کند، اما قادر است مقدار درهم کامل را به طور قابل اعتماد بازسازی نماید.

در «طیف دقت»، فیلتر خارج‌قسمت جایی بین جدول درهم و فیلتر بلوم قرار دارد. اگر دو مقدار کلید متمایز به یک اثرانگشت یکسان درهم شوند، فیلتر خارج‌قسمت قادر به تشخیص تفاوت بین آن‌ها به شیوه‌ای که جدول درهم‌سازی می‌تواند، نخواهد بود. اما اگر دو مقدار کلید به اثرانگشت‌های متفاوتی درهم شوند، فیلتر خارج‌قسمت قادر خواهد بود آن‌ها را از یکدیگر تشخیص دهد. این وضعیت در مورد فیلترهای بلوم صادق نیست، جایی که پرس و جو بر روی مقدار کلیدی با مجموعه‌ای منحصربه‌فرد از k درهم ممکن است مثبت کاذب ایجاد نماید. فیلتر خارج‌قسمت کارکردی مشابه فیلتر بلوم اما با طراحی متفاوت دارد. با استفاده از اثرانگشت‌های طویل‌تر، فیلترهای خارج‌قسمت می‌توانند میزان مثبت کاذب را کاهش دهند، اما اثرانگشت‌های طویل‌تر ممکن است فضای زیادی نیز مصرف کنند.

در این بخش، صناعات و ترفندهای مختلفی را خواهیم دید که فیلتر خارج‌قسمت برای ذخیره‌سازی فشرده و موجز اثرانگشت‌ها به کار می‌برد. قابلیت بازسازی اثرانگشت کامل زمانی مفید واقع می‌شود که بخواهیم مقدارها را حذف کنیم. به دیگر سخن، فیلترهای خارج‌قسمت می‌توانند به طور کارآمد حذف را انجام دهند.

مضافا، فیلتر خارج‌قسمت قادر به تغییر اندازه است و ادغام دو فیلتر خارج‌قسمت در یک فیلتر خارج‌قسمت بزرگ‌تر عملیاتی نسبتا ساده، حافظ مقادیر، و سریع است. ادغام کارآمد، تغییر اندازه کارآمد، و حذف کارآمد همگی ویژگی‌هایی هستند که ممکن است منجر به استفاده از فیلتر خارج‌قسمت به جای استفاده از فیلتر بلوم در برخی کاربردها شود. این ویژگی‌ها به ویژه در سیستم‌های توزیع شده پویا بسیار مفید هستند.

از طرف دیگر، به دلیل نیاز به دسترسی تصادفی چندباره به هر عملیات، کارائی فیلتر بلوم در صورت عدم گنجایش در حافظه اصلی و نیاز به حافظه جانبی سخت افت می‌کند و نامناسب می‌شود.

در ادامه، طراحی فیلتر خارج‌قسمت را بررسی می‌کنیم. درک و پیاده‌سازی فیلترهای خارج‌قسمت نسبت به فیلترهای بلوم اندکی پیچیده‌تر است. اما سعی می‌شود در فرایندی گام به گام مفاهیم و مراحل فیلتر خارج‌قسمت بیان شود. ابتدا با توضیح خارج‌قسمت‌گیری آغاز و سپس با توصیف این که چگونه فیلتر خارج‌قسمت از بیت‌های متاداده همراه با خارج‌قسمت‌گیری برای صرفه‌جویی در فضا استفاده می‌کند، ادامه می‌دهیم. یکی از راه‌های به نظر آوردن فیلتر خارج‌قسمت، مشاهده آن به عنوان بازی برای صرفه‌جویی یک بیت اینجا و آنجا، و به کارگیری ترفندهای مختلف در این راستاست. فیلتر خارج‌قسمت تنها داده‌ساختار از این نوع نیست، اما برخی از ترفندهایی که در آن استفاده می‌شود می‌توانند به طور کلی در درک داده‌ساختارهای مشابه مبتنی بر جدول‌های درهم فشرده مفید واقع شوند.

تمرین- چه ابزارهای دیگری در عمل شبیه فیلتر خارج‌قسمت کار می‌کنند.

خارج‌قسمت‌گیری

خارج‌قسمت‌گیری مقدار درهم ورودی را به دو بخش خارج‌قسمت و باقیمانده تقسیم می‌کند. در فیلتر خارج‌قسمت از خارج‌قسمت برای نمایه‌گذاری در سطل متناظر جدول درهم استفاده می‌شود و باقیمانده همان چیزی است که در خانه متناظر ذخیره می‌گردد. برای مثال با فرض درهم به طول p بیت و اندازه جدول m معادل ۲ به توان q، خارج‌قسمت مقداری است که با q بیت چپ درهم تعیین می‌شود و باقیمانده نشان‌دهنده r بیت باقیمانده به تعداد p - q بیت است.

مثال ۱۱- با فرض m=۳۲ و p=۱۱ باشد، آن‌گاه q=۵ و r=۶. حتی در این مثال خرد، به دلیل خارج‌قسمت‌گیری به ازای هر خانه معادل $q = ۵$ بیت صرفه‌جویی می‌شود.

سخن کوتاه، جهت بهبود محلی‌سازی فضایی، داده‌ساختار فیلتر خارج قسمت با جدول درهم فشرده با آدرس‌دهی باز با $m = ۲^q$ سطل نمایش داده می‌شود که در آن باقیمانده f_r در سطلی که با خارج‌قسمت f_q نمایه شده است ذخیره می‌شود. تصادم‌های احتمالی با کاوش خطی حل می‌شوند.

شبه‌کد زیر روش خارج‌قسمت‌گیری را روشن می‌سازد.

الگوریتم ۷- خارج‌قسمت‌گیری مقدار درهم

ورودی: اثر انگشت یا درهم f

خروجی: خارج‌قسمت f_q و باقیمانده f_r

$f_r \leftarrow f \% ۲^r$

$f_q \leftarrow \left\lfloor \frac{f}{۲^r} \right\rfloor$

برگرداندن f_q و f_r

قطعه کد زیر نشان می‌دهد پس از درهم شدن مقدار کلید و ذخیره شدن درهم در متغیر اثرانگشت، فرایند درج در فیلتر خارج‌قسمت چگونه پیش می‌رود.

p = len(f) ⟵ تعداد بیت‌های در درهم یا اثر انگشت

q = log2(m) ⟵ فرض بر اندازه درهم توانی از دو

r = p – q

f_q = math.floor(f / 2 ** r) ⟵ q بیت سمت چپ اثرانگشت

f_r = f % 2 ** r) ⟵ r بیت سمت راست اثرانگشت

filter(f_q) = rem ⟵ ذخیرهٔ باقی‌مانده در سطل متناظر خارج قسمت

اگر هیچ دو اثرانگشتی دارای خارج‌قسمت یکسان نباشند به معنای عدم تصادم است. در این صورت، هر باقیمانده سطل مخصوص به خود یعنی b را اشغال می‌کند. بازسازی درهم کامل به سادگی از طریق الحاق بازنمایی دودویی شماره سطل b با بازنمایی دودویی باقیمانده ذخیره شده در سطل b امکان‌پذیر است. به دیگر سخن، با توجه به باقیمانده f_r در سطل f_q، اثر انگشت کامل را می‌توان به طور منحصربه‌فرد به صورت زیر بازسازی کرد:

$$f = f_q \times ۲^r + f_r$$

این ویژگی به تغییر اندازه و ادغام کمک می‌کند اما باید توجه داشت مقدار اصلی (کلید یا مقداری که با اعمال تابع درهم‌ساز مقدار درهم متناظر حساب می‌شود) را نمی‌توان بازسازی کرد. بار دیگر تأکید می‌شود که فیلتر خارج‌قسمت جایی بین جدول‌های درهم‌سازی قرار داریم که مقدارهای کلید واقعی را نگهداری می‌کنند و فیلترهای بلوم که تناظر بین هر کلید و مقادیر متناظر درهم‌ها را نمی‌توانند بازسازی کنند.

با این حال تصادم در جدول‌های درهم بسیار رایج است و فیلترهای خارج‌قسمت تصادم‌ها را با استفاده از گونه‌ای از کاوش خطی حل می‌کنند. از هم اکنون می‌توان حدس زد که پیچیدگی‌هایی رخ می‌دهد. زیرا در نتیجه کاوش خطی برخی از باقیمانده‌ها از سطل اصلی خود به سمت خانه‌های بعدی رانده می‌شوند و بنابراین ارتباط بین خارج‌قسمت و باقیمانده از دست می‌رود. برای بازسازی درهم کامل، فیلتر خارج‌قسمت به ازای هر خانه از سه بیت متاداده اضافی استفاده می‌کند. سه بیت در مقابل صرفه‌جویی حدود ۲۰ تا ۳۰ بیت به ازای هر خانه بابت خارج‌قسمت در جدول‌های درهم‌سازی بزرگ‌تر بهائی اندک است. در ادامه، چگونگی تاثیر بیت‌های متاداده در تسهیل عملیات در فیلتر خارج‌قسمت را توضیح می‌دهیم.

بیت‌های مَتاداده

پیش از توضیح دقیق نقش بیت‌های متاداده، چند اصطلاح را معرفی می‌کنیم.

مفهوم اجرا یا Run: هر اجرا توالی پیوسته‌ای از خانه‌های فیلتر خارج‌قسمت است که در آن‌ها باقی‌مانده‌ها (یا همان اثرانگشت‌ها) با یک خارج‌قسمت یکسان ذخیره شده‌اند. به بیان دیگر، همهٔ اثرانگشت‌هایی که روی مدخلی خاص تصادم دارند، «پشت‌سرهم» و به ترتیب صعودیِ باقی‌مانده‌ها ذخیره می‌شوند. به‌علت وقوع تصادم‌ها و عمل هل دادن یا انتقال هنگام درج، ممکن است اجرا از فاصله‌ای بسیار دورتر از مدخل سطل متناظر خود آغاز شود.

مفهوم خوشه: خوشه رشته‌ای پیوسته و بدون مدخلی مقدار نگرفته از یک یا چند اجرا است. این رشته شامل خانه‌هایی است که همگی باقیمانده در خود دارند، و اولین باقیمانده موجود دقیقاً در خانهٔ اصلیِ درهم‌شدهٔ خود قرار گرفته است. این خانه، مدخل لنگر و تکیه نام دارد. پایان خوشه یا با مدخلی خالی، یا با شروع خوشه‌ای دیگر مشخص می‌شود.

هنگام انجام عملیات جست‌وجو در فیلتر باقیمانده نیاز است باقیمانده‌ها را همراه با بیت‌های متاداده تفسیر کنیم تا بتوان اثرانگشت کامل را بازیابی کرد. تفسیر همیشه از آغاز خوشه (یعنی موقعیت لنگر) شروع می‌شود و رو به خانه‌های بعدی ادامه می‌یابد. سه بیت متادادهٔ موجود در هر خانه چنین نقش‌هایی دارند:

۱. bucket_occupied: این بیت درهم‌شدن کلیدی روی سطل متناظر را مشخص می‌کند. اگر مقدار آن ۱ باشد: حداقل یک کلید به این سطل درهم شده است. اگر ۰ باشد: هرگز کلیدی به این سطل درهم نشده است. سخن کوتاه، این بیت روشن می‌کند کدام خارج‌قسمت‌ها ممکن است در خوشه وجود داشته باشند.

۲. run_continued: این بیت باقیمانده‌های فعلی ادامهٔ اجرای قبلی است یا آغاز اجرای جدید را مشخص می‌کند. اگر مقدار آن ۰ باشد: باقیمانده فعلی اولین عضو اجرا است. اگر ۱ باشد: باقیماندهٔ فعلی به اجرای قبلی تعلق دارد. پس، این بیت مرزبندی دقیق اجراها را مشخص می‌کند.

۳. is_shifted: این بیت حضور باقیمانده موجود در خانهٔ اصلی متناظر یا جابجایی آن به دلیل تصادم را مشخص می‌کند. اگر مقدار آن ۰ باشد: باقیمانده در خانهٔ اصلی درهم‌شده خود قرار دارد. اگر ۱ باشد: باقیمانده به‌سمت خانه‌های بعدی منتقل شده است. این بیت کمک می‌کند آغاز خوشه را بیابیم.

هر سطل شامل سه بیت متادادهٔ bucket _occupied، run_continued، و is_shifted است که همگی در ابتدا بدون مقدارند و نقشی مهم در مدیریت داده‌ساختار ایفا می‌کنند.

پس هر مدخل دارای نشانی و نمایهٔ متناظر خارج‌قسمت، و ذخیرهٔ مقدار باقیمانده و متاداده‌ها جهت وصل باقیمانده و خارج‌قسمت متناظر با یک مقدار درهم است. شکل زیر موارد مذکور را نشان می دهد

شکل ۱- سطل یا مدخل در فیلتر خارج‌قسمت

سطل f_q

Is_shifted منتقل شده یا سرجای خود	Run_continued ادامه اجرا	Bucket_occupiedاشغال سطل
f_r		

درج در فیلتر خارج‌قسمت

جهت فهم بهتر در ابتدا مثالی را درباره افزودن مقادیری به فیلتر خارج‌قسمت پیش می‌بریم.

مثال ۱۲-فرایند درج مقدارهای v، w و x را گام به گام دنبال می‌کنیم. با فیلتر خارج‌قسمت در ابتدا خالی با ۳۲ خانه برابر با ۲ به توان ۵ شروع می‌کنیم که در آن هر خانه و همچنین هر سه بیت متاداده ابتدا روی مقدار ۰ تنظیم شده است:

۱- درج v: مقدار درهم h(v) برابر با ۱۰۰۰۱۰۰۱۰۱۰ است. سطلی که با خارج‌قسمت ۱۰۰۰۱ تعیین می‌شود قبلاً اشغال نشده است. بیت bucket_occupied را روی مقدار ۱ تنظیم و باقیمانده را در خانه‌ای که با خارج‌قسمت آن تعیین می‌شود ذخیره می‌شود. نیازی به هیچ اقدام اضافی روی سایر بیت‌ها نیست زیرا این مورد در حال حاضر هم آغاز یک اجرا و هم آغاز یک خوشه محسوب می‌شود.

۲- درج w: مقدار درهم h(w) برابر با ۱۰۰۱۱۱۰۰۱۱۱ است. مجدداً بیت bucket_occupied را در خارج‌قسمت ۱۰۰۱۱ روی مقدار ۱ تنظیم می‌کنیم و باقیمانده را در خانه متناظر در دسترس ذخیره می‌نماییم و هیچ بیت دیگری را تغییر نمی‌دهیم.

خارج‌قسمت/نشانی	باقیمانده	متاداده		
۱۰۰۰۰				
۱۰۰۰۱	۰۰۱۰۱۰	۱	۰	۰
۱۰۰۱۰				
۱۰۰۱۱	۱۰۰۱۱۱	۱	۰	۰
۱۰۱۰۰				
۱۰۱۰۱				
۱۰۱۱۰				
۱۰۱۱۱				

سطل اشغال شده: مقداری به این سطل درهم شده است
اجرا ادامه‌دار: مقدار فعلی ادامه اجرایی است
منتقل شده: این مدخل انتقالی است

۳- درج x: مقدار درهم h(x) برابر با ۱۰۰۱۱۱۱۱۱۱۱ است. سطل متناظر با خارج‌قسمت ۱۰۰۱۱ هنگامی که سعی می‌کنیم بیت را روی مقدار ۱ تنظیم کنیم از پیش اشغال شده است. بنابراین هر جا که باقیمانده را ذخیره کنیم باید بیت run_continued آن خانه را روی مقدار ۱ تنظیم کنیم. خانه در سطل درهم‌شده اشغال می‌باشد. از این رو هر جا که باقیمانده را ذخیره کنیم باید بیت is_shifted آن خانه را نیز روی مقدار ۱ تنظیم نماییم. با توجه به این که در ابتدای خوشه‌ای قرار داریم که فقط یک اجرا دارد (که خارج‌قسمت آن با خارج‌قسمت x برابر است)، به سمت پایین پویش می‌کنیم تا نخستین خانه در دسترس را درون اجرا در سطل ۱۰۱۰۰ پیدا کنیم. باقیمانده را ذخیره می‌کنیم و بیت‌های run_continued و is_shifted را روی مقدار ۱ تنظیم می‌نماییم.

خارج‌قسمت/نشانی	باقیمانده	متاداده		
۱۰۰۰۰				
۱۰۰۰۱	۰۰۱۰۱۰	۱	۰	۰
۱۰۰۱۰				
۱۰۰۱۱	۱۰۰۱۱۱	۱	۰	۰
۱۰۱۰۰	۱۱۱۱۱۱	۰	۱	۱
۱۰۱۰۱				
۱۰۱۱۰				
۱۰۱۱۱				

سطل اشغال شده: مقداری به این سطل درهم شده است
اجرا ادامه‌دار: مقدار فعلی ادامه اجرایی است
منتقل شده: این مدخل انتقالی است

در حال حاضر فیلتر خارج‌قسمت دارای دو خوشه است که هر خوشه دارای یک اجرا است. در ادامه، چند مقدار دیگر درج می‌کنیم.

۴- درج y: مقدار درهم h(y) برابر با ۱۰۱۰۰۱۰۱۱۰۱ است. بیت bucket_occupied در خارج‌قسمت ۱۰۱۰۰ مقدار صفر داشت و ما آن را روی مقدار ۱ تنظیم می‌کنیم (بیت run_continued در خانه باقیمانده نهایی روی مقدار ۰ تنظیم خواهد شد) و خانه در سطل درهم‌شده اشغال می‌باشد (بیت is_shifted در خانه باقیمانده نهایی روی مقدار ۱ تنظیم خواهد شد). با شروع از ابتدای خوشه

به دنبال نخستین مکان برای ذخیره اجرای جدید خود می‌گردیم. نخستین خانه در دسترس در خارج‌قسمت ۱۰۱۰۱ قرار دارد. از این رو باقیمانده را ذخیره کرده و بیت‌های متاداده را متناسب با آن تنظیم می‌کنیم.

خارج‌قسمت/نشانی	باقیمانده	متاداده		
۱۰۰۰۰				
۱۰۰۰۱	۰۰۱۰۱۰	۱	۰	۰
۱۰۰۱۰				
۱۰۰۱۱	۱۰۰۱۱۱	۱	۰	۰
۱۰۱۰۰	۱۱۱۱۱۱	۱	۱	۱
۱۰۱۰۱	۱۰۱۱۰۱	۰	۰	۱
۱۰۱۱۰				
۱۰۱۱۱				

سطل اشغال شده: مقداری به این سطل درهم شده است
اجرا ادامه‌دار: مقدار فعلی ادامه اجرایی است
منتقل شده: این مدخل انتقالی است

۵- درج z: مقدار درهم h(z) برابر با ۱۰۱۰۰۱۱۱۱۱۰ است. بیت bucket_occupied سطل مربوط به z از پیش روی مقدار ۱ تنظیم شده است (بیت run_continued در خانه نهایی روی مقدار ۱ خواهد بود) و خانه اصلی اشغال است (بیت is_shifted در خانه نهایی روی مقدار ۱ خواهد بود). با شروع از ابتدای خوشه محل اجرای مناسب خود را پیدا می‌کنیم. در امتداد اجرا به سمت پایین تا خارج‌قسمت ۱۰۱۱۰ پویش می‌کنیم تا باقیمانده z را ذخیره کنیم و بیت‌ها را متناسب با آن تنظیم می‌نماییم.

خارج‌قسمت/نشانی	باقیمانده	متاداده		
۱۰۰۰۰				
۱۰۰۰۱	۰۰۱۰۱۰	۱	۰	۰
۱۰۰۱۰				
۱۰۰۱۱	۱۰۰۱۱۱	۱	۰	۰
۱۰۱۰۰	۱۱۱۱۱۱	۱	۱	۱
۱۰۱۰۱	۱۰۱۱۰۱	۰	۰	۱
۱۰۱۱۰	۱۱۱۱۱۰	۰	۱	۱
۱۰۱۱۱				

اجرا
اجرا
خوشه

سطل اشغال شده: مقداری به این سطل درهم شده است
اجرا ادامه‌دار: مقدار فعلی ادامه اجرایی است
منتقل شده: این مدخل انتقالی است

به دلیل ترتیب صعودی مقادیر درج شده، عمل درج نسبتاً ساده شده است. ترتیب مرتب شده درج‌ها در فیلتر خارج‌قسمت پیرنگی ممکن است اما غالب نیست همچنین، باید توانایی مدیریت پیرنگ‌هائی ممکن باشد که در آن اثرانگشت‌های درج شده به ترتیب دلخواه و نامرتب می‌رسند.

۶- زمانی که مقدارها خارج از ترتیب مرتب شده اثرانگشت درج می‌شوند، پس از آن که اجرای صحیح برای باقیمانده‌ای که قرار است در آن درج شود پیدا شد، ممکن است آن درج چندین مورد از آن اجرا و سایر مقدارهای موجود در آن خوشه را به خانه‌های بعدی جابجا کند. نمونه درج یک مقدار با نام a با مقدار درهم h(a) برابر با ۱۰۱۰۰۰۰۰۰۰ را در فیلتر خارج‌قسمت حاصل از شکل اخیر در نظر بگیرید. این مقدار متعلق به دومین اجرای دومین خوشه خواهد بود که در حال حاضر خانه‌های تعیین شده با خارج‌قسمت‌های ۱۰۱۰۱ و ۱۰۱۱۰ را اشغال کرده است. مقدار a کل اجرا را یک خانه به سمت پایین جابجا می‌کند تا بتواند در خانه ۱۰۱۰۱ ذخیره شود زیرا باقیمانده آن نخستین مقدار در ترتیب صعودی مرتب شده در آن اجرا محسوب می‌شود و در نتیجه همچنین باعث ایجاد تغییرات در بیت‌های متاداده می‌گردد. باقیمانده‌ها باید در درون یک اجرا مرتب شده باشند و همچنین اجراها در میان خوشه‌ها مرتب باشند؟ پاسخ به این پرسش زمانی مطرح خواهد شد که درباره تغییر اندازه کارآمد و ادغام کارآمد صحبت می‌کنیم.

خارج‌قسمت/نشانی	باقیمانده	متاداده		
۱۰۰۰۰				
۱۰۰۰۱	۰۰۱۰۱۰	۱	۰	۰
۱۰۰۱۰				
۱۰۰۱۱	۱۰۰۱۱۱	۱	۰	۰
۱۰۱۰۰	۱۱۱۱۱۱	۱	۱	۱
۱۰۱۰۱	۰۰۰۰۰۰	۰	۰	۱
۱۰۱۱۰	۱۰۱۱۰۱	۰	۱	۱
۱۰۱۱۱	۱۱۱۱۱۰	۰	۱	۱

اجرا
اجرا
خوشه

سطل اشغال شده: مقداری به این سطل درهم شده است
اجرا ادامه‌دار: مقدار فعلی ادامه اجرایی است
منتقل شده: این مدخل انتقالی است

نکته مهم دیگر مرتبط با جدول درهم این است که نیاز بالقوه به جابجایی کل یک خوشه از مقادیر در حین درج یا حذف و همچنین رمزگشایی کل یک خوشه در حین انجام عملیات جستجو بر اهمیت کوچک بودن اندازه خوشه‌ها تأکید می‌کند. هر چه فضای خالی بیشتری باقی بماند، احتمال تشکیل خوشه‌های بزرگ که عملیات درج و جستجو نیاز به پویش و رمزگشایی در آن‌ها داشته باشد کاهش می‌یابد. درست مانند جدول‌های درهم معمولی با کاوش خطی، فیلترهای خارج‌قسمت زمانی سریع‌تر کار می‌کنند که ضریب بار در محدوده ۷۵ تا ۹۰ درصد نگه داشته شود.

تمرین- مقادیر ضریب جهت کار مناسب را تحقیق و تحلیل کنید.

الگوریتم‌های عملیات های فیلتر خارج‌قسمت

هنگامی که دو اثر انگشت متفاوت f و f' خارج‌قسمتی یکسان دارند ($f_q = f'_q$)، تصادمی نرم رخ داده است که می‌توان با کاوش خطی حل کرد. در فیلتر خارج‌قسمت با ذخیره تمام باقیمانده‌های اثر انگشت‌ها با خارج‌قسمت یکسان به طور پیوسته در اجرا پیاده‌سازی می‌شود. در صورت لزوم، باقیمانده می‌تواند از مکان اصلی خود به جلو منتقل شود و در سطل بعدی ذخیره شود. اگر به انتهای آرایه برسیم نیاز است آرایه را دوری دید و حلقه زد و به ابتدای آرایه برگشت.

در ادامه الگورتیم را با توجه به مثال ۱۲ معرفی می‌کنیم. گاهی اوقات برای افزودن باقیمانده‌ای جدید نیاز به جابجائی به راست مقادیر موجود و ایجاد مدخلی خالی داریم. الگوریتم زیر این فرایند را نشان می‌دهد.

الگوریتم ۸- استفاده از جابجایی به راست جهت تخلیهٔ سطل‌ها		
۱	ورودی: نمایه سطل i	
۲	ورودی: فیلتر خارج‌قسمت به طول m	
۳	$qbl \leftarrow QF[i]$	
۴	$k \leftarrow i + 1$	
۵	تازمانی که درست (True) انجام بده	
۶	اگر $k \geq m$ آنگاه	
۷	$k \leftarrow 0$	
۸	اگر $QF[k] = \phi$ آنگاه	
۹	$QF[k] \leftarrow qbl$	
۱۰	$QF[k].\text{run\_continued} \leftarrow 1$	هر مقدار منتقل شده ادامه اجراست
۱۱	$QF[i] \leftarrow \phi$	در موقعیت اصلی خود نیست
۱۲	$QF[i].\text{bucket\_occupied} \leftarrow 0$	
۱۳	برگرداندن QF	
۱۴	ورنه	
۱۵	$knoni \leftarrow QF[k]$	
۱۶	$QF[k] \leftarrow qbl$	
۱۷	$QF[k].\text{run\_continued} \leftarrow 1$	تبدیل به ادامهٔ اجرا
۱۸	$QF[k].\text{is\_shifted} \leftarrow 1$	منتقل شده
۱۹	$qbl \leftarrow knoni$	
۲۰	$k \leftarrow k + 1$	

دنباله‌ای از یک یا چند اجرای متوالی بدون سطل خالی در بین آنها، خوشه نامیده می‌شود. همه خوشه‌ها بلافاصله پیش از سطل خالی قرار می‌گیرند و بیت is_shifted اولین مقدار آن هرگز مقداردهی نمی‌شود.

جدول درهم داخلی به صورت فشرده در آرایه‌ای ذخیره می‌شود تا حافظه مورد نیاز کاهش یابد و محلی‌سازی بهتری حاصل شود. با این حال، این امر پیچیده‌تر شدن مدیریت را موجب می‌شود. تابع پیمایشی را در نظر می‌گیریم که برای یافتن اجرا طراحی شده است. گام‌برداری عقب‌رو از سطل متناظرِ f برای یافتن ابتدای خوشه شروع می‌شود. به محض پیدا شدن شروع خوشه، دوباره به جلو گام برمی‌دارد تا مکان اولین باقیمانده برای سطل f_q را پیدا کند، که شروع واقعی اجرا $r_{\text{آغاز}}$ است.

الگوریتم ۹- پیمایش فیلتر خارج‌قسمت برای یافتن اجرا		
۱	ورودی: اندیس سطل متعارف f_q، فیلتر خارج‌قسمت	
۲	$j \leftarrow f_q$	یافتن آغاز خوشه دارای f_q
۳	تازمانی که $QF[j].is\ shifted == 1$ انجام بده	
۴	$j--$	

```
۵   r_آغاز ← j
۶   تازمانی که j ≠ f_q
۷       تازمانی که ۱ == QF[r_آغاز + ۱].runContinued
۸           r_آغاز + +
۹       r_آغاز + +
۱۰      تازمانی که ۱ ≠ QF[j + ۱].bucktOccupied
۱۱          j + +
۱۲      j + +
۱۳  r_آغاز ← r_پایان                                   یافتن پایان اجرای دارای f_q
۱۴  تازمانی که ۱ == QF[r_پایان + ۱].runContinued & r_پایان + ۱ ≤ m
۱۵      r_پایان + +
۱۶  برگرداندن r_آغاز و r_پایان
```

خط ۱ ورودی‌های الگوریتم را مشخص می‌کند. در خط دوم متغیر j مقداردهی اولیه می‌شود تا از f_q به سمت چپ حرکت کند. j نقش اشاره‌گر به هر اجراست که بر اساس آن ابتدا و و انتهای خوشه تعیین می‌شود. خط ۳ و ۴ در پی یافتن ابتدای خوشه هستند. با یافتن آن،در خط پنجم ابتدای اجرا مقداردهی اولیه می‌شود. در ادامه در حلقه خط ۶ تا۱۲ به دنبال ابتدای اجرای متناظر f_q است. از خط شش به بعد j نقش شمارندهٔ اجراها را دارد و هر تکرار یک اجرا را به اتمام می‌رساند. پس برای هر اجرا ابتدا و انتهای آن تعیین و جستجو ادامه می‌یابد تا به اجرای متناظر f_q برسد. حلقه‌های داخلی این وظیفه را بر عهده دارند. حلقه خطوط ۷ و ۸ ابتدای اجرا را تا زمانی که اجرا ادامه دارد به سمت چپ هدایت می‌کند. در انتهای حلقه متغیر $r_{\text{آغاز}}$ به آخرین عضو اجرای فعلی اشاره می‌کند. خط ۹ متغیر $r_{\text{آغاز}}$ را متناظر اولین مقدار اجرای بعدی می‌کند. به دیگر سخن اولین حلقه وظیفه نادیده‌گرفتن اجراها تا رسیدن به ابتدای اجرای متناظر ورودی را دارد. وظیفه حلقه دوم افزایش مقدار j و در نتیجه تا زمانی است که سطلی اشغال نشده باشد. به بیان دیگر j را متناظر با اجرایی می‌کند که اغازش در متغیر $r_{\text{آغاز}}$ ذخیره شده است. با برابر شدن j با مقدار خارج قسمت، به اجرای مدنظر رسیده‌ایم و خود بخود متغیر $r_{\text{آغاز}}$ به ابتدای آن اشاره می‌کند. خطوط ۱۳ تا ۱۵ پایان اجرای مدنظر را پیدا می‌کند و اینکار با بررسی اینکه مقدار بعدی متغیر ادامه اجرا برابر صفر است افزایش می‌یابد. با صفر شدن مقدار ادامه اجرا بعدی به معنای به انتهای اجرای مدنظر رسیدن است. در نهایت مقادیر حاصل متناظر برگردانده می‌شود.

هنگامی که می‌خواهیم مقداری جدید را درج کنیم، ابتدا خارج‌قسمت و باقیمانده آن را محاسبه می‌کنیم. اگر سطل متناظر برای مقدار اشغال نشده باشد، بلافاصله با استفاده از روش درج در الگوریتم زیر وارد می‌شود. در غیر این صورت، قبل از درج، لازم است سطلی مناسب با تابع پیمایش از الگوریتم بالا پیدا شود. هنگامی که سطل صحیح پیدا شد، درج واقعی همچنان نیاز به ادغام مناسب f_r در دنباله مقادیر ذخیره شده قبلی دارد که ممکن است نیاز به جابجایی به راست مقادیر بعدی و به‌روزرسانی بیت‌های متاداده متناظر داشته باشد. با استراتژی انتخاب مذکور برای سطل مناسب و تابع جابجایی به راست که در الگوریتم ۸ داده شده است، روش کامل درج به صورت زیر تدوین می‌شود.

الگوریتم۱۰- درج مقدار در فیلتر خارج‌قسمت

```
۱   ورودی: عضو x ∈ D
۲   ورودی: فیلتر خارج‌قسمت با تابع درهم‌ساز h
```

۳	$f \leftarrow h(x)$	
۴	$f_q, f_r \leftarrow f$	
۵	اگر $QF[f_q].\text{bucket}Occupied == ۰$ و خالی بودن $QF[f_q]$	
۶	$QF[f_q] \leftarrow f_r$	خالی و اشغال نبودن سطل
۷	$QF[f_q].\text{bucket}Occupied \leftarrow ۱$	
۸	برگرداندن True	
۹	$QF[f_q].\text{bucket}Occupied \leftarrow ۱$	علامت اشغال شدن سطل متناظر
۱۰	$r_{\text{آغاز}}, r_{\text{پایان}} \leftarrow Scan(QF, f_q)$	
۱۱	برای i از $r_{\text{آغاز}}$ تا $r_{\text{پایان}}$	
۱۲	اگر $QF[i] == f_r$	به معنای وجود f_r از قبل و لزوم خروج
۱۳	برگرداندن True	
۱۴	وگر $QF[i] > f_r$ آنگاه	درج f_r در سطل i و جابجا کردن بقیه
۱۵	$QF \leftarrow ShiftRight(QF, i)$	
۱۶	$QF[i] \leftarrow f_r$	
	اگر $i \neq r_{\text{آغاز}}$	
۱۷	$QF[i].run\ continued \leftarrow ۱$	اولین نبودن در اجرا
۱۸	اگر $i \neq f_q$	
۱۹	$QF[i].is\_shifted \leftarrow ۱$	
۲۰	برگرداندن True	
۲۱	$QF \leftarrow ShiftRight\left(QF, r_{\text{پایان}} + ۱\right)$	
۲۲	$QF\left[r_{\text{پایان}} + ۱\right] \leftarrow f_r$	باقیمانده جدید بزرگتر از تمامی مقادیر فعلی در اجرا، درج در انتهای اجرا و بسط همان اجرا
۲۳	$QF[i].run\ continued \leftarrow ۱$	
۲۴	اگر $r_{\text{پایان}} + ۱ \neq f_q$	
۲۵	$QF[r_{\text{پایان}} + ۱].is\_shifted \leftarrow ۱$	
۲۶	برگرداندن True	

با توجه به طرح کاوش خطی، طول اکثر اجراها از مرتبهٔ $O(۱)$ است و نویسندگان فیلتر اشاره کردند که به احتمال زیاد همه اجراها دارای طول $O(\log m)$ هستند.

تمرین- چرا طول اجراها محتملا برابر $O(\log m)$ است؟

مثال ۱۳- افزودن مقادیر به فیلتر: فیلتر خارج‌قسمت با اثر انگشت‌های ۱۶ بیتی حاصل از نسخهٔ ۳۲ بیتی تابع درهم‌ساز مورمور۳ را در نظر می‌گیریم:

$$h(x) = MurmurHash3(x)\%\ ۱۶$$

جهت سطل‌بندی، $q = ۳$ بیت با بیشترین اهمیت را رزرو می‌کنیم، و تبعا اندازهٔ فیلتر برابر $m = ۲^{۳} = ۸$ است و $p = ۱۳$ بیت باقیمانده را در سطل‌های انتخاب شده ذخیره می‌کنیم.

همانند مثال ۳ و ۴، نمایه‌سازی نام پایتخت‌ها را پیش می‌بریم و کپنهاگ را در ابتدا به فیلتر می‌افزائیم. اثر انگشت آن را با استفاده از تابع درهم‌ساز h عبارت است از:

$$f = h(Copenhagen) = ۴۲۴۸۲۲۴۲۰۷$$

با توجه به الگوریتم ۷، خارج‌قسمت و باقیمانده عبارتند از:

$$f_q = \left\lfloor \frac{f}{۲^{۱۳}} \right\rfloor = ۷, f_r = f \% ۲^{۱۳} = ۴۹۰۱۲۷۸۲۳.$$

سطل متعارف برای مقدار کپنهاگ $j = f_q = ۷$ است که می‌خواهیم باقیمانده f_r را در آن اندیس کنیم. به دلیل اشغال نبودن سطل‌ها درجی به راحتی صورت می‌پذیرد و $f_r = ۴۹۰۱۲۷۸۲۳$ را در سطل با اندیس ۷ درج و بیت bucket_occupied را برابر یک قرار می‌دهیم:

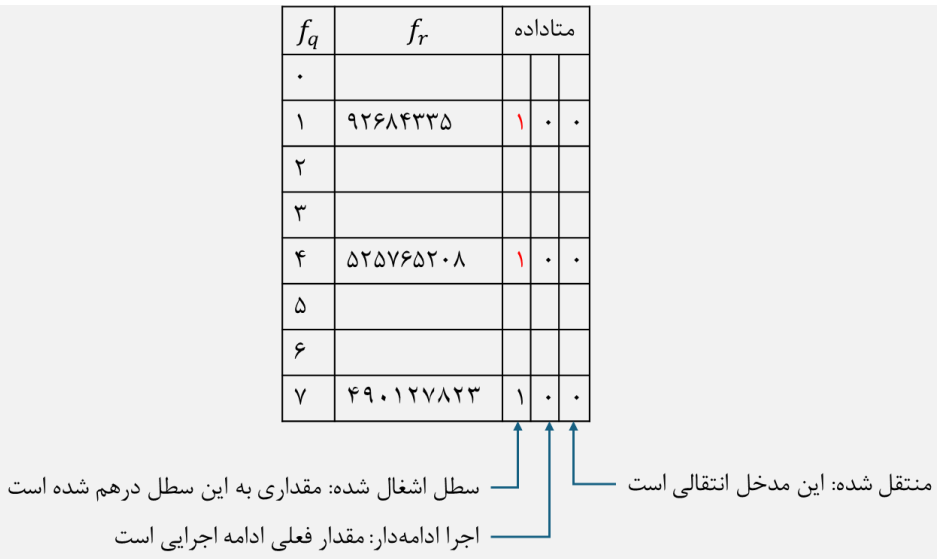
f_q	f_r	متاداده		
۰				
۱				
۲				
۳				
۴				
۵				
۶				
۷	۴۹۰۱۲۷۸۲۳	۱	۰	۰

منتقل شده: این مدخل انتقالی است

سطل اشغال شده: مقداری به این سطل درهم شده است

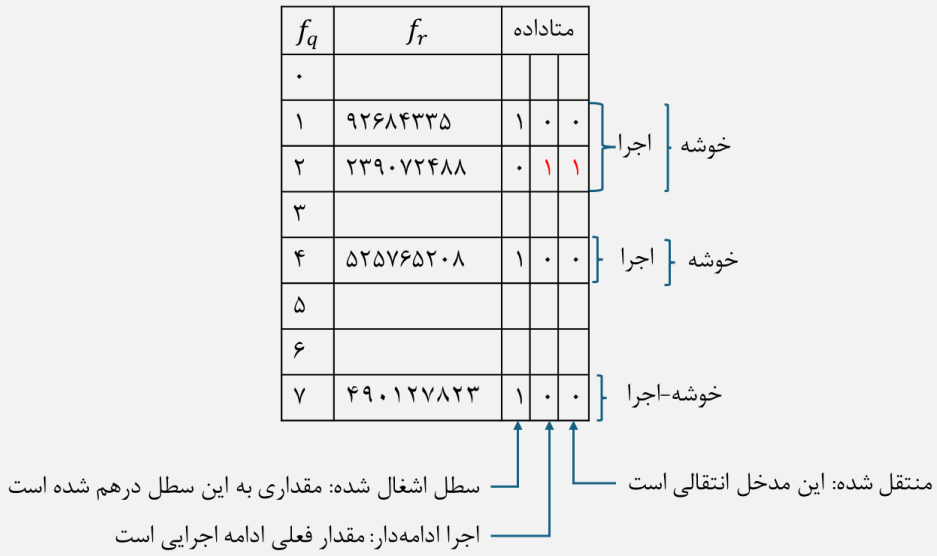
اجرا ادامه‌دار: مقدار فعلی ادامه اجرایی است

به همین ترتیب، لیسبون دارای اثر انگشت f = ٦٢٩٥٥٥٢٤٧ و سطل متناظر ۱، و پاریس با اثر انگشت f = ٢٦٧٣٢٤٨٨٥٦ و سطل متناظر ۴، را اندیس می‌کنیم. به دلیل آزاد بودن سطل‌های متناظر، باقیمانده‌ها را به ترتیب درج می‌کنیم و بیت‌های bucket_occupied را مقدار می‌دهیم:

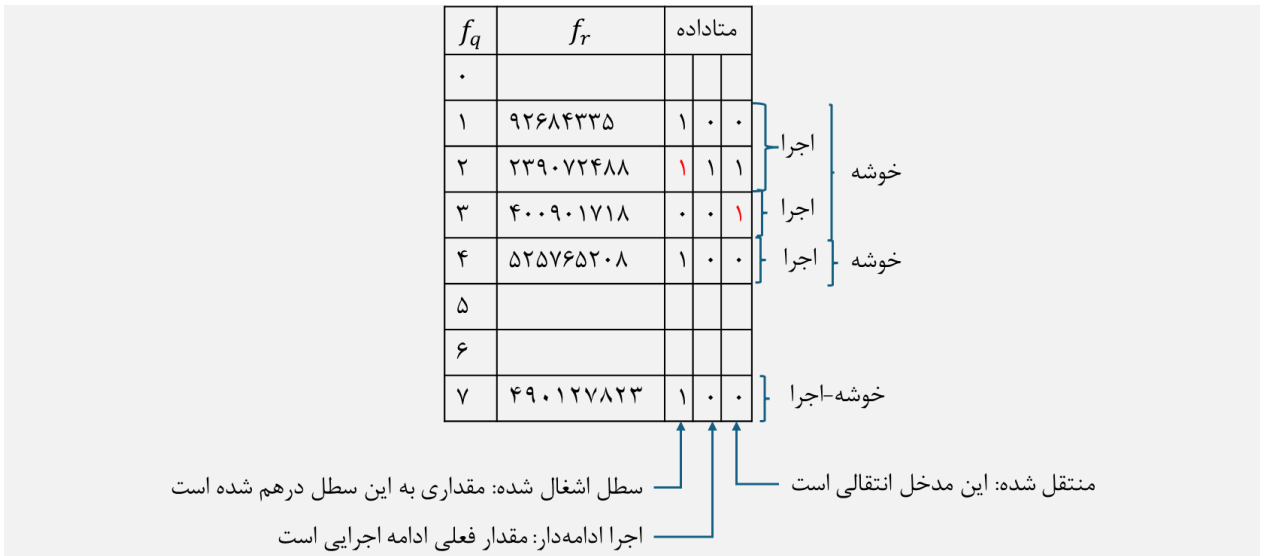


سپس، استکهلم را با اثر انگشت f = ۷۷۵۹۴۳۴۰۰ اضافه می‌کنیم و سطل متعارف آن $j = f_q =$ ۱ و باقیماندهٔ $f_r =$ ۲۳۹۰۷۲۴۸۸ را به دست می‌آوریم. با این حال، سطل متعارف ۱ قبلاً بیت bucket_occupied آن مقداردهی شده است که به معنای این است که قبلاً با باقیمانده مقدار دیگری (در این مورد مقدار لیسبون) اشغال شده است.

چون بیت‌های is_shifted و run_continued تنظیم نشده‌اند، پس در ابتدای خوشه هستیم که در عین حال شروع اجرا نیز است. باقیمانده f_r بزرگتر از مقدار اندیس شده قبلی ۹۲۶۸۴۳۳۵ است، بنابراین باید در سطل بعدی موجود، یعنی سطل ۲ ذخیره شود و بیت‌های is_shifted و run_continued آن باید تنظیم شوند. با این حال، بیت bucket_occupied برای سطل ۲ بدون تغییر باقی می‌ماند، زیرا دارای باقیمانده ذخیره‌ای متناظر با سطل متعارفش نیست.



ورودی بعدی زاگرب است که اثر انگشت f = ۱۴۷۴۶۴۳۵۴۲، سطل متناظر j = ۲ و باقیمانده $f_r =$ ۴۰۰۹۰۱۷۱۸ دارد. اما، همانطور که بیت مقداردهی شده is_shifted نشان می‌دهد سطل ۲ قبلاً با مقداری انتقالی استفاده شده است، ولی bucket_occupied تنظیم نشده است. بنابراین، مقدار f_r نیز باید به سمت راست، به سطل بعدی موجود که در این مورد سطل ۳ است منتقل شود.

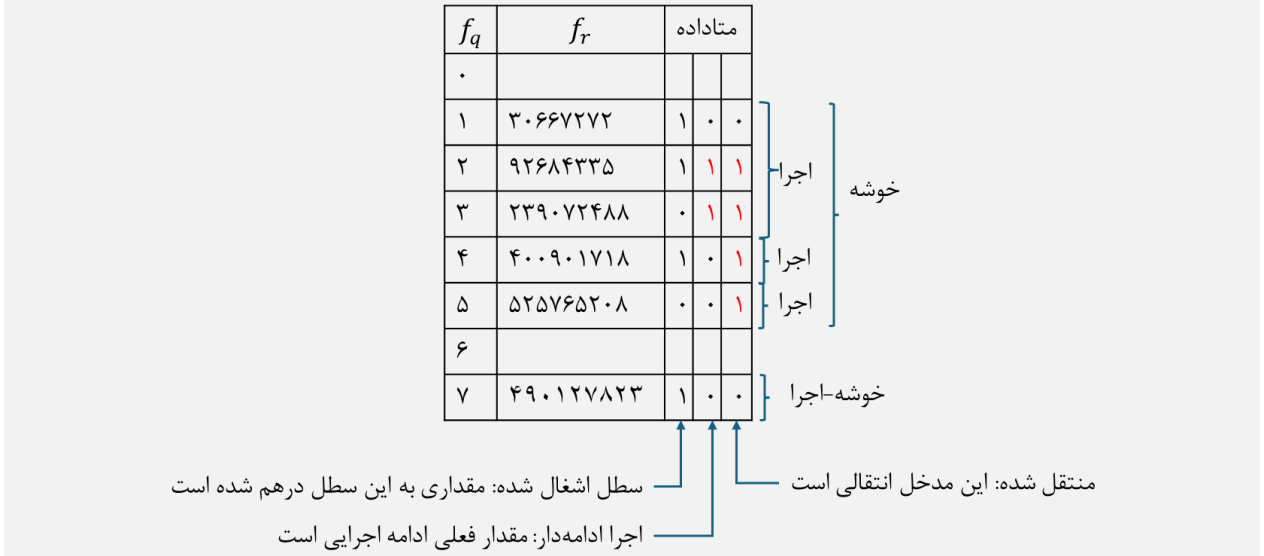


بیت is_shifted مقداردهی می‌شود تا نشان دهد سطل حاوی مقداری منتقل شده از موقعیت متعارف خود است، بیت run_continued به دلیل متناظر بودن اولین عضو آن با سطل متناظر تغییر داده نمی‌شود. مضافا، bucket_occupied سطل ۲ جهت به یاد داشتن اینکه حداقل یک باقیمانده در سطل متعارفش ذخیره شده مقداردهی می‌شود.

در پایان، ورشو را با اثر انگشت $f = ۵۶۷۵۳۸۱۸۴$ اضافه می‌کنیم که خارج قسمت و باقیماندهٔ آن برابر است با

$$f_q = \left\lfloor \frac{f}{۲^{۱۳}} \right\rfloor = ۱$$
$$f_r = f \% ۲^{۱۳} = ۳۰۶۶۷۲۷۲$$

سطل متعارف $j = f_q = ۱$ بر اساس بیت bucket_occupied اشغال شده است. هر چند، بیت‌های دیگر مقداری ندارند و در نتیجه به معنای در ابتدای خوشه بودن است. مقدار باقیمانده f_q از مقدار اندیس شدهٔ ۹۱۶۸۴۳۳۵ کوچکتر است، در نتیجه باید در سطل متعارف اندیس شود، و بقیهٔ باقیمانده‌ها منتقل شوند و به عنوان ادامه مقداردهی شوند.

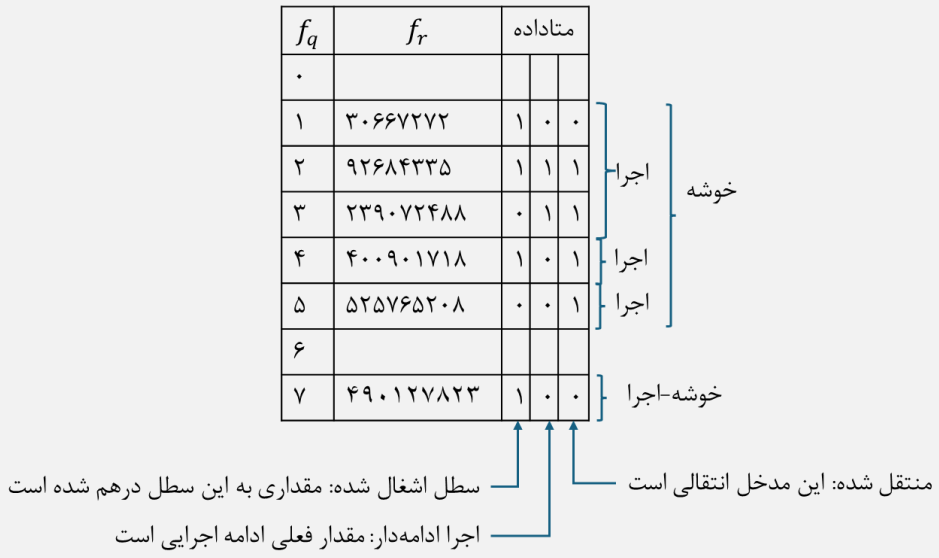


آزمون عنصرها همانند درج انجام می‌شود. سطل متعارف برای عضو بررسی می‌شود تا وجود باقیمانده متناظر در فیلتر با مشاهده بیت bucket_occupied ببینند. در صورت یک نبودن مقدار بیت مذکور، نتیجه می‌گیریم مقدار قطعا در فیلتر نیست. وگرنه، فیلتر را رویه اسکن بررسی می‌کنیم تا اجرای مناسب سطل یافت شود. سپس، در اجرا، باقیمانده‌های حاضر را با باقیمانده مقدار مقایسه می‌کنیم. اگر یافت شد، احتمال وجود آن در فیلتر برگردانده می‌شود. الگوریتم آزمون عضویت به صورت زیر است:

الگوریتم۱۱- آزمون عضویت در فیلتر خارج‌قسمت

۱	ورودی: مقدار $x \in D$	
۲	ورودی: فیلتر خارج‌قسمت با تابع درهم‌ساز h	
۳	خروجی: False اگر مقدار یافت نشد و True اگر احتمال حضور وجود داشته باشد	
۴	$f \leftarrow h(x)$	
۵	$f_q, f_r \leftarrow f$	
۶	اگر $QF[f_q].bucket\_occupied \neq ۱$ آنگاه	
۷	برگرداندن False	
۸	در غیر این صورت	
۹	$r_{\text{آغاز}}, r_{\text{پایان}} \leftarrow Scan(QF, f_q)$	
۱۰	برای i از $r_{\text{آغاز}}$ تا $r_{\text{پایان}}$ انجام بده	جستجوی f_r در داخل اجرا
۱۱	اگر $QF[i] == f_r$ آنگاه	
۱۲	برگرداندن True	
۱۳	برگرداندن False	

مثال- آزمون عضویت در فیلتر- داده‌ساختار حاصل مثال قبلی را در نظر می‌گیریم.



پاریس را با خارج‌قسمت ۴ $= f_q$ و باقیمانده ۵۲۵۷۶۵۲۰۸ $= f_r$ می‌آزمائیم. سطل ۴ از قبل اشغال شده است، به این معنی که حداقل یک باقیمانده در جایی از فیلتر وجود دارد که آن را به عنوان سطل متناظر دارد. با این وجود فعلا نمی‌توان مقدار سطل را با f_r مقایسه کرد، زیرا بیت is_shifted مقدار یک دارد و در نتیجه باید در پی اجرایی بود که با سطل متناظر ۴ در خوشه فعلی مطابقت دارد. بنابراین، از سطل ۴ به سمت چپ پیمایش می‌کنیم و سطل‌ها را با مقدار بیت‌های bucket_occupied می‌شماریم تا به شروع خوشه برسیم. در این مثال، خوشه از سطل ۱ شروع می‌شود و دو سطل اشغال شده (سطل‌های ۱ و ۲) در سمت چپ سطل ۴ قرار دارند. بنابراین، اجرای ما سومین در خوشه است و باید از ابتدای خوشه (سطل ۱) به سمت راست اسکن کنیم تا با شمارش سطل‌هایی که بیت‌های run_continued آنها تنظیم نشده است، به آن اجرا برسیم. در نهایت، متوجه می‌شویم که اجرا از سطل ۵ شروع می‌شود و مقدار را با باقیمانده‌های ذخیره مقایسه با در نظر گرفتن اینکه آنها به ترتیب صعودی مرتب شده‌اند مقایسه می‌کنیم. مقدار موجود در سطل ۵ دقیقاً با باقیمانده ۵۲۵۷۶۵۲۰۸ $= f_r$ مطابقت دارد، بنابراین می‌توانیم نتیجه بگیریم که مقدار پاریس ممکن است در فیلتر وجود داشته باشد.

حذف از فیلتر خارج‌قسمت به روشی سخت شبیه به افزودن مقدار جدید انجام می‌شود. با این حال، چون تمام باقیمانده‌های اثر انگشت‌ها با خارج‌قسمت یکسان با توجه به ترتیب عددی خود به صورت پیوسته ذخیره می‌شوند، حذف یک باقیمانده از خوشه باید تمام اثر انگشت‌ها را جابجا کند تا ورودی خالی پس از حذف پر شود و بیت‌های متاداده را به ترتیب اصلاح کند.

الگوریتم۱۲- استفاده از انتقال به چپ برای پر کردن سطل‌های خالی	
۱	ورودی: اندیس سطل i
۲	ورودی: فیلتر خارج‌قسمت به طول m
۳	$k \leftarrow i + ۱$
۴	تازمانی که $QF[k] \neq \emptyset$ انجام بده
۵	$QF[k - ۱] \leftarrow QF[k]$
۶	$QF[k - ۱].\text{is\_shifted} \leftarrow QF[k].is\_shifted$
۷	$QF[k] \leftarrow \phi$
۸	$QF[k].run\_continued \leftarrow ۰$
۹	$QF[k].\text{is\_shifted} \leftarrow ۰$
۱۰	$k \leftarrow k + ۱$
۱۱	اگر $k > m$ آنگاه
۱۲	$k \leftarrow ۰$

ابتدا، لازم است بررسی کنیم که آیا سطل متعارف از قبل اشغال شده است، در غیر این صورت مقدار قطعاً در فیلتر نیست و می‌توانیم در اینجا متوقف شویم. پس از آن از روش اسکن برای یافتن سطل مناسب و حذف مقدار درخواستی (در صورت وجود) استفاده می‌کنیم و مقادیر بعدی را جابجا کرده و بیت‌های متاداده متناظر را به‌روزرسانی می‌کنیم. لازم به توجه است، اگر باقیمانده حذف شده آخرین باقیمانده برای سطل متعارف خود بود، بیت bucket_occupied را نیز پاک می‌کنیم.

الگوریتم ۱۳- حذف مقدار از فیلتر خارج‌قسمت	
۱	ورودی: مقدار $x \in D$
۲	ورودی: فیلتر خارج‌قسمت با تابع درهم‌ساز h

۳	خروجی: False اگر مقدار یافت نشد و True در غیر این صورت	
۴	$f \leftarrow h(x)$	
۵	$f_q, f_r \leftarrow f$	
۶	اگر ۱ $\neq QF[f_q].bucket\_occupied$ آنگاه برگرداندن True	
۷	$r_{\text{آغاز}}, r_{\text{پایان}} \leftarrow Scan(QF, f_q)$	
۸	برای i از $r_{\text{آغاز}}$ تا $r_{\text{پایان}}$ انجام بده	
۹	اگر f_r == QF[i] آنگاه	یافتن مقدار و حذف آن
۱۰	$QF[i] \leftarrow \phi$	
۱۱	$QF[i].run\_continued \leftarrow$ ۰	
۱۲	$QF[i].isShifted \leftarrow$ ۰	
۱۳	اگر $i <$ $r_{\text{پایان}}$ آنگاه	انتقال به چپ
۱۴	$QF \leftarrow ShiftLeft(QF, i)$	
۱۵	اگر $r_{\text{پایان}} ==$ $r_{\text{آغاز}}$ آنگاه	تهی شدن اجرا
۱۶	$QF[i].bucket\_occupied \leftarrow$ ۰	صفر شدن مقدار اشغال
۱۷	برگرداندن True	
۱۸	برگرداندن False	نبودن عضو در اجرا

فیلتر خارج‌قسمت می‌تواند اثر انگشت‌ها را از داده‌های ذخیره شده بازیابی کند، بنابراین از حذف، ادغام، و تغییر اندازه پشتیبانی می‌کند. ادغام بر نسبت مثبت کاذب فیلترها تأثیر نمی‌گذارد. همچنین، برخلاف فیلتر بلوم شمارنده که فقط از حذف‌های احتمالی درست پشتیبانی می‌کند حذف در فیلتر خارج‌قسمت همیشه درست است.

تغییر اندازه فیلتر خارج‌قسمت (هم کوچک‌سازی و هم بزرگ‌سازی) را می‌توان با تکرار روی فیلتر و کپی کردن هر اثر انگشت در داده‌ساختاری تازه تخصیص یافته بدون نیاز به درهم‌سازی مجدد انجام داد. دو یا چند فیلتر خارج‌قسمت را می‌توان با استفاده از الگوریتمی شبیه به مرتب‌سازی ادغامی، الگوریتم مرتب‌سازی تقسیم و غلبه ابداعیِ جان فون نویمان ادغام کرد. بنابراین، همه فیلترهای ورودی را می‌توان به صورت موازی اسکن کرد و نتیجه ادغام شده در فیلتر خروجی نوشته می‌شود. زمان مورد نیاز برای انجام یک آزمایش، افزودن یا حذف در یک فیلتر خارج‌قسمت تحت سلطه زمان اسکن به عقب و جلو است. با این حال، فیلتر خارج‌قسمت با تمرکز بر کلان‌داده (به عنوان مثال، ۱ میلیارد مقدار برای تابع درهم‌ساز ۶۴ بیتی) طراحی شده است و برای مجموعه‌های داده کوچک یا متوسط، پیچیدگی آن می‌تواند مزایا را کاهش دهد.
تصویر زیر نحوهٔ تغییر اندازه و افزایش اندازه فیلتر خارج‌قسمت را نشان می‌دهد. جهت دو برابر کردن فضا،کافی است یک بیت از باقیمانده سمت چپ‌ترین بیت حذف و به سمت راست خارج‌قسمت افزوده شود.

متاداده			باقیمانده	خق
				۰۰۰۰
				۰۰۰۱
				۰۰۱۰
				۰۰۱۱
۰	۰	۱	۱۰۱۰	۰۱۰۰
				۰۱۰۱
				۰۱۱۰
				۰۱۱۱
				۱۰۰۰
				۱۰۰۱
				۱۰۱۰
				۱۰۱۱
				۱۱۰۰
				۱۱۰۱
۰	۱	۱	۰۱۱۱	۱۱۱۰
				۱۱۱۱

متاداده			باقیمانده	خق
				۰۰۰
				۰۰۱
۰	۰	۱	۰۱۰۱۰	۰۱۰
				۰۱۱
				۱۰۰
				۱۰۱
				۱۱۰
۰	۰	۱	۰۰۱۱۱	۱۱۱

متاداده			باقیمانده	خق
				۰۰
۰	۰	۱	۰۰۱۰۱۰	۰۱
				۱۰
۰	۰	۱	۱۰۰۱۱۱	۱۱

تمرین- الگوریتم ادغام دو فیلتر خارج‌قسمت را بنویسید.

فیلتر خارج‌قسمت دارای ویژگی‌هایی است که در ادامه تشریح می‌شود.

الف- مثبت کاذب امکان‌پذیر است. داده‌ساختار فیلتر خارج‌قسمت نمایش فشرده‌ای از یک مجموعه چندگانه از اثر انگشت‌ها است و نرخ مثبت کاذب آن تابعی از تابع درهم‌ساز h و تعداد مقادیر n اضافه شده به فیلتر است. مضافا، دو مقدار متفاوت می‌توانند مقادیر یکسانی برای باقیمانده و خارج‌قسمت داشته باشند، که به آن تصادم سخت می‌گویند. به دلیل چنین رویدادهای بسیار نادری، امکان وقوع پاسخ‌های مثبت کاذب وجود دارد و احتمال آنها pr_{fp} با کران بالای زیر محدود می‌شود:

$$pr_{fp} \approx ۱ - e^{-\frac{n}{۲^p}} \leq \frac{n}{۲^p}$$

معادله بالانشان می‌دهد که با تعداد ثابت مقادیر مورد انتظار n، بین احتمال مثبت کاذب pr_{fp} و طول اثر انگشت p می‌توان سبک سنگینی کرد. پیاده‌سازی‌های عملی فیلترهای خارج‌قسمت از اثر انگشت‌های ۳۲ و ۶۴ بیتی استفاده می‌کنند.

ضریب بار در فیلتر خارج‌قسمت همانند سایر جدول‌های درهم‌ساز، سخت مهم است و در پی تخصیص حداقل اندازه به تعداد مقادیر مورد انتظار سطل هستیم، به این معنی که تعداد سطل‌ها m را به صورت زیر انتخاب می‌کنیم:

$$m = ۲^q > n$$

و طول باقیمانده r را می‌توان از (۲.۶) به صورت زیر محاسبه کرد:

$$r = \left\lceil \log\left(-\frac{n}{۲^q} \times \frac{۱}{\ln(۱ - pr_{fp})} \right) \right\rceil$$

ب- منفی کاذب امکان‌پذیر نیست. مانند سایر داده‌ساختارهای فصل جاری، اگر فیلتر خارج‌قسمت عنصری را عضو نداند، قطعاً عضو مجموعه نیست:

$$pr_{fn} = ۰$$

با وجود اینکه فیلتر خارج‌قسمت حدود بیست درصد از فیلتر بلوم بزرگتر، در عین حال سریع‌تر است. زیرا هر دسترسی فقط به ارزیابی یک تابع درهم‌ساز نیاز دارد و همه داده‌ها در بلوک‌های پیوسته ذخیره می‌شوند. آزمون‌ها در فیلتر خارج‌قسمت منجر به یک عدم وجود در حافظه کش می‌شود، در مقابل حداقل به طور میانگین دو عدم وجود در حافظه کش برای الگوریتم فیلتر بلوم است.

میزان مثبتِ کاذب و ملاحظات فضایی

وقوع مثبت کاذب در فیلتر خارج‌قسمت زمانی است که دو کلید متفاوت اثرانگشت یکسان تولید کنند. بر اساس تحلیل ارائه‌شده در منابع، اگر جدول شامل 2^q خانه باشد و طول اثرانگشت برابر $p = q + r$ باشد، آنگاه احتمال وقوع مثبتِ کاذب تقریباً برابر با $\approx 2^{-r}$ است. یعنی این احتمال اساساً به تعداد بیت‌های باقی‌مانده وابسته است. فضای موردنیاز برای جدول فیلتر خارج‌قسمت برابر با $2^q(r + 3)$ بیت است. تعداد عناصری که در فیلتر درج می‌شوند برابر با $n = \alpha 2^q$ است که در آن α ضریب بارگذاری است. این ضریب تأثیر چشمگیری بر کارایی عملیات درج، جست‌وجو و حذف دارد.
گونهٔ کم‌حجم‌تر از فیلتر خارج‌قسمت نیز وجود دارد که تنها از دو بیت متاداده استفاده می‌کند. این نسخه به فضای $2^q(r + 2)$ بیت نیاز دارد و با وجود کاهش یک بیت، میزان مثبتِ کاذب را افزایش نمی‌دهد. با این‌حال، این نسخه مرحلهٔ رمزگشایی را به‌طور قابل توجهی پیچیده می‌سازد. به‌ویژه در حضور خوشه‌های بلند، که در این حالت عملیات متداول (درج، جست‌وجو و حذف) به‌شکل چشمگیری وابسته به پردازنده و پرهزینه می‌شوند.
مقایسهٔ کاربردی با فیلترهای بلوم: در عمل، به‌سببِ فضای اضافی لازم برای جدولِ جست‌وجوی خطی، برای میزان رایجِ مثبتِ کاذب فیلترهای خارج‌قسمت معمولاً اندکی بیش از فیلترهای بلوم فضا مصرف می‌کنند. بااین‌حال، برای میزان بسیار پایینِ مثبتِ کاذب، فیلترهای خارج‌قسمت نسبت به فیلترهای بلوم از کارایی فضایی بیشتری برخوردارند.
نتایج مقایسه عملکرد در حافظه در [Be۱۲] نشان می‌دهد که فیلترهای خارج‌قسمت می‌توانند ۲.۴ میلیون درج در ثانیه را مدیریت کنند در حالی که فیلترهای بلوم به حدود ۰.۶۹ میلیون محدود هستند. با این حال، با آزمون برای عناصر، آنها تقریباً در همان سطح حدود ۲ میلیون در ثانیه هستند.

مثال ۲.۱۱: حافظه مورد نیاز: همانطور که در معادله طول باقیمانده بیان شد، برای مدیریت ۱ میلیارد عنصر، فیلتر خارج‌قسمت باید حداقل 2^{30} سطل داشته باشد، در نتیجه نمی‌توان از اثر انگشت‌های کوتاه‌تر از ۳۱ بیت استفاده کرد. اگر بخواهیم احتمال رویدادهای مثبت کاذب را حدود دو درصد نگه داریم، تعداد بیت‌ها برای باقیمانده را می‌توان به صورت زیر پیدا کرد:

$$r = \left\lceil \log\left(-\frac{10^9}{2^{30}} \times \frac{1}{\ln(1 - 0.02)} \right) \right\rceil = 6$$

بنابراین، طول مورد نیاز اثر انگشت $p = q + r = 36$ بیت است، که ۳۰ بیت اول برای سطل‌بندی استفاده می‌شوند و ۶ بیت باقی‌مانده در سطل مناسب ذخیره می‌شوند. از آنجایی که هر سطل علاوه بر این شامل سه بیت متاداده است، اندازه کل فیلتر خارج‌قسمت $2^{30} \times 9$ بیت معادل تقریباً ۱.۲ گیگابایت حافظه است.

فیلتر فاخته

بیشتر اصلاحات فیلتر بلوم کلاسیک که از حذف پشتیبانی می‌کنند، از نظر فضا یا عملکرد دچار افت می‌شوند. برای حل این مشکل، بین فن، دیوید اندرسون، مایکل کامینسکی و مایکل میتزن ماخر در سال ۱۳۹۳ فیلتر فاخته را پیشنهاد کردند که نوعی فشرده از

جدول درهم فاخته بود که بالاتر معرفی شد، اما، به جای ذخیرهٔ جفت‌های کلید-مقدار، اثر انگشت‌هایی با طول p برای هر مقدار درج شده تطبیق می‌یابد.

فیلترهای فاخته دارای پیاده‌سازی آسان‌تر، پشتیبان درج و حذف پویا، و در عین حال در بسیاری از کاربردهای عملی فضای کمتری نسبت به سایر اصلاحات فیلتر بلوم استفاده می‌کنند و حتی عملکرد بالاتری نیز از خود بروز می‌دهند.

داده‌ساختار CuckooFilter با جدول درهم‌ساز فاخته چندگانه با m سطل، که هر کدام تا b مقدار را ذخیره می‌کند، نمایش داده می‌شود. در درهم‌ساز فاخته استاندارد، برای درج عنصری جدید، لازم است به مقادیر موجود اصلی دسترسی باشد تا مشخص کنیم در صورت نیاز به فضا برای مقادیر جدید، مقادیر ذخیره شده را کجا منتقل کنیم. با این حال، فیلتر فاخته فقط اثر انگشت‌ها را ذخیره می‌کند و راهی برای بازیابی مقادیر اصلی و درهم‌سازی مجدد آنها برای یافتن سطل جدیدشان در جدول درهم‌ساز وجود ندارد.

با هدف غلبه بر این محدودیت و همچنان استفاده از درهم‌ساز فاخته، الگوریتم فیلتر فاخته از «درهم‌ساز فاخته با کلید جزئی» استفاده می‌کند، که استخراج مقدار موجود را از سطل جدید مقدار بدون دانستن خود مقدار اصلی و صرفا از اثر انگشت آن ممکن می‌کند. بر اساس طرح مذکور، هر مقدار x که درج می‌شود، الگوریتم اثر انگشت p-بیتی f و اندیس‌های دو سطل نامزد را به صورت زیر محاسبه می‌کند:

$$i = h(x)\%m$$
$$j = (i \oplus (h(f)\%m)\%m$$

هنگامی که طول اثر انگشت p در مقایسه با طول فیلتر m کوچک است، عملیات یای انحصاری فقط تعداد کمی از بیت‌های پایین‌تر را تغییر می‌دهد، اما بیشتر بیت‌های مرتبه بالاتر ثابت می‌مانند. این نشان می‌دهد که مقادیر انتقال یافته از سطل‌های اصلی خود تمایل به قرارگیری در نزدیکی یکدیگر در سطل‌های جایگزین دارند و در نتیجه توزیع در جدول درهم‌ساز دچار انحراف می‌شود که بر کارایی فیلتر تأثیر می‌گذارد. درهم‌ساز اثر انگشت‌ها تضمین می‌کند که این مقادیر به سطل‌هایی در بخش‌های کاملاً متفاوت جدول درهم‌ساز منتقل می‌شوند، بنابراین تصادم‌ها درهم‌ساز را کاهش می‌دهد و استفاده از جدول را بهبود می‌بخشد.

عملیات OR انحصاری $\oplus$ (XOR) در معادله این ویژگی مهم را تضمین می‌کند که با دانستن سطل‌های فعلی مقدار k می‌توان سطل جایگزین آن k^* را بدون بازیابی مقدار اصلی محاسبه کرد:

$$k^* = \big(k \oplus h(f)\big)\%m$$

برای درج مقدار جدید x به فیلتر فاخته، اندیس‌های دو سطل نامزد را با معادلات بالا محاسبه می‌کنیم. در صورت خالی بودن یکی از دو سطل حاصل، مقدار در آن درج می‌شود. در غیر این صورت، به طور تصادفی یکی از آن سطل‌ها را انتخاب می‌کنیم و مقدار x را در آنجا ذخیره می‌کنیم، در حالی که مقدار را از آن سطل به سطل نامزد جایگزین آن با استفاده از (۲.۱۱) منتقل می‌کنیم. این روش را تا زمانی که یک سطل خالی پیدا شود یا تا زمانی که به حداکثر تعداد جابجایی‌ها برسیم، تکرار می‌کنیم. اگر سطل خالی وجود نداشته باشد، فیلتر پر در نظر گرفته می‌شود.

الگوریتم ۲.۱۴: درج مقدار در فیلتر فاخته
ورودی: مقدار $x \in D$
ورودی: فیلتر فاخته با اثر انگشت‌گذاری و تابع درهم‌ساز h
خروجی: True اگر مقدار اضافه شده باشد و False در غیر این صورت
$f \leftarrow$ انگشت اثر (x)
$i \leftarrow h(x)$
$j \leftarrow i \oplus h(f)$

اگر CUCKOOFILTER [i] فضای خالی دارد آنگاه
CUCKOOFILTER [i].add (f)
برگرداندن True
در غیر این صورت اگر CUCKOOFILTER [j] فضای خالی دارد آنگاه
CUCKOOFILTER [j] .add (f)
برگرداندن True
$k \leftarrow$ نمونه $(\{i, j\})$
برای n از صفر تا بیش-تکرار انجام بده
$x \leftarrow$ نمونه$(CUCKOOFILTER\ [k])$
f و اثر انگشت ذخیره شده در ورودی x را با هم عوض کن
$k = k \oplus h(f)$
اگر CUCKOOFILTER [k] فضای خالی دارد آنگاه
$CUCKOOFILTER\ [k]\ .add\ (f)$
برگرداندن True
برگرداندن False

مثال ۲.۱۲- افزودن مقادیر به فیلتر

داده‌ساختار CUCKOOFILTER به طول $m = ۱۰$ را در نظر می‌گیریم که برای سادگی، فقط یک اثر انگشت $p = ۱۶$ بیتی در هر سطل ذخیره می‌کند. از یک تابع درهم‌ساز MurmurHash۳ با اندازه ۳۲ بیتی برای محاسبهٔ اثر انگشت‌ها و اندیس‌های سطل استفاده می‌کنیم. مانند مثال‌های دیگر، نام پایتخت‌ها را درج می‌کنیم، با مقدار کپنهاگ شروع می‌کنیم که اثر انگشت p-بیتی آن است.

$$f = MurmurHash3(Copenhagen)\%\ ۲^p = ۴۹۶۱۵$$

هنگامی که سطل اصلی i طبق معادلهٔ اصلی است:

$$i = MurmurHash3(Copenhagen)\%\ m = ۷$$

و سطل جایگزین j را می‌توان از i و اثر انگشت f به صورت زیر استخراج کرد:

$$j = \left(i \oplus MurmurHash3(f)\right)\%\ m = (۷ \oplus ۳۴۴۷۵۵۴۵)\%\ ۱۰ = ۰$$

بنابراین، می‌توانیم اثر انگشت f را در سطل ۷ یا ۰ اندیس کنیم، و از آنجایی که فیلتر خالی است، از سطل اصلی استفاده می‌کنیم:

۰	۱	۲	۳	۴	۵	۶	۷	۸	۹
							۴۹۶۱۵		

به طور مشابه، مقدار آتن را با اثر انگشت $f = ۲۷۳۵۶$ و سطل‌های نامزد ۰ و ۷ اندیس می‌کنیم. سطل اصلی ۰ اشغال نشده است و اجازه می‌دهد اثر انگشت آزادانه ذخیره شود:

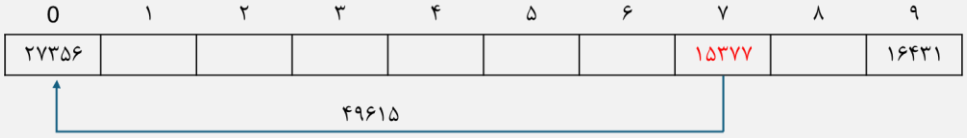
۰	۱	۲	۳	۴	۵	۶	۷	۸	۹
۲۷۳۵۶							۴۹۶۱۵		

مقدار لیسبون را در نظر می‌گیریم که اثر انگشت آن $f = ۱۶۴۳۱$ و سطل‌های نامزد ۷ و ۹ هستند. ما با سطل اصلی ۷ شروع می‌کنیم، اما قبلاً اشغال شده است و در حداکثر ظرفیت یک است، بنابراین سطل جایگزین ۹ را بررسی می‌کنیم که خالی است و اثر انگشت را در آنجا ذخیره می‌کنیم:

۰	۱	۲	۳	۴	۵	۶	۷	۸	۹
۲۷۳۵۶							۴۹۶۱۵		۱۶۴۳۱

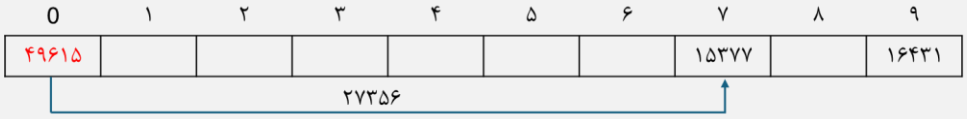
سپس، مقدار هلسینکی را در نظر می‌گیریم. دارای اثر انگشت f = ۱۵۳۷۷ و هر دو اندیس سطل برابر با ۷ هستند. لازم به توجه است، چنین تصادم اندیسی برای فیلترهای کوچک، مانند این مثال، از فیلترهای واقعی محتمل‌تر است. سطل ۷ اشغال شده است و نمی‌تواند بیش از یک مقدار را بپذیرد، بنابراین باید روش جابجایی را در فیلتر شروع کنیم. ابتدا، با سطل k شروع می‌کنیم و مقدار ۴۹۶۱۵ را از سطل ۷ با مقدار f مبادله می‌کنیم، سپس آن مقدار را به یک سطل جدید k که از آن با معادله (۲.۱۱) استخراج می‌شود، منتقل می‌کنیم:

$$k = (۷ \oplus MurmurHash3(۴۹۶۱۵))\%\ ۱۰ = ۰$$



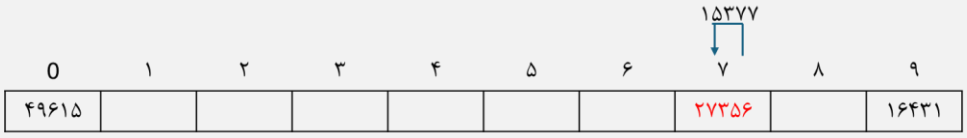
متأسفانه، سطل ۰ قبلاً حاوی مقدار ۲۷۳۵۶ است و ما آن را با ۴۹۶۱۵ مبادله می‌کنیم و باید اندیس سطل جدید را برای آن محاسبه کنیم:

$$k = (۰ \oplus MurmurHash3(۲۷۳۵۶))\%\ ۱۰ = ۷$$



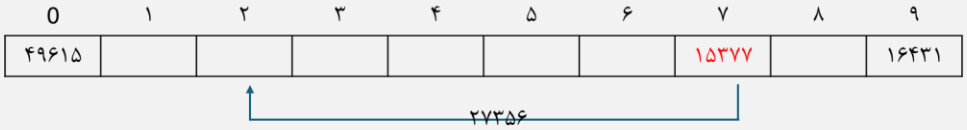
ما به سطل ۷ برگشتیم که خالی نیست و باید روش جابجایی را یک بار دیگر تکرار کنیم. ابتدا، مقدار ۲۷۳۵۶ را در سطل ذخیره می‌کنیم، و سپس یک سطل جدید برای مقدار ۱۵۳۷۷ محاسبه می‌کنیم:

$$k = (۷ \oplus \text{MurmurHash۳}(۱۵۳۷۷))\%\ ۱۰ = ۷$$

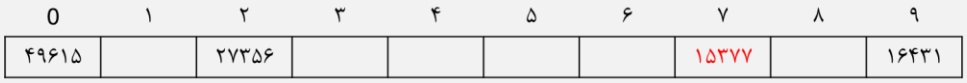


به دلیل تصادم اندیسی که قبلاً برای آن اثر انگشت ذکر کردیم، دوباره به سطل ۷ برمی‌گردیم و مقدار ۱۵۳۷۷ را در آن ذخیره می‌کنیم در حالی که مقدار ۲۷۳۵۶ را به یک سطل جدید k منتقل می‌کنیم:

$$k = (۷ \oplus MurmurHash3(۲۷۳۵۶))\%\ ۱۰ = ۲$$



سطل ۲ خالی است، بنابراین می‌توان مقدار ۲۷۳۵۶ را ذخیره و درج را به پایان رساند.



آزمون وجود مقدار در فیلتر ساده است. ابتدا، برای مقدار مورد آزمایش، اثر انگشت و سطل‌های نامزد آن را محاسبه می‌کنیم. اگر اثر انگشت در هر یک از سطل‌ها وجود داشته باشد، نتیجه می‌گیریم که مقدار ممکن است وجود داشته باشد. در غیر این صورت، قطعاً در فیلتر نیست.

الگوریتم ۲.۱۵: آزمون مقدار در فیلتر فاخته

ورودی: مقدار $x \in D$

ورودی: فیلتر فاخته با اثر انگشت‌گذاری و تابع درهم‌ساز h

خروجی: False اگر مقدار یافت نشد و True اگر ممکن است وجود داشته باشد

$f \leftarrow$ انگشت اثر (x)

$i \leftarrow h(x)$

$j \leftarrow i \oplus h(f)$

اگر $f \in CuckooFilter[i]$ یا $f \in CuckooFilter[j]$ آنگاه

برگرداندن True

برگرداندن False

مثال ۲.۱۳: آزمون مقادیر در فیلتر

داده‌ساختار CuckooFilter را که در مثال ۲.۱۲ ساختیم در نظر می‌گیریم:

0	1	2	3	4	5	6	7	8	9
۴۹۶۱۵		۲۷۳۵۶					۱۵۳۷۷		۱۶۴۳۱

بیایید مقدار لیسبون را که سطل‌های نامزد آن ۷ و ۹ هستند و اثر انگشت آن $f = ١٦٤٣١$ همانطور که در بالا محاسبه کردیم، آزمایش کنیم. می‌توانیم مقدار ۱۶۴۳۱ را در سطل ۹ پیدا کنیم و نتیجه بگیریم که مقدار لیسبون ممکن است در فیلتر وجود داشته باشد. در مقابل، مقدار اسلو را در نظر می‌گیریم که دارای اثر انگشت $f = ٥٣١٠٤$ و سطل‌های نامزد ۰ و ۶ است. همانطور که می‌بینیم، چنین مقداری در این سطل‌ها وجود ندارد، بنابراین مقدار اسلو قطعاً در فیلتر نیست.

برای حذف عنصر، اثر انگشت را ساخته، سپس اندیس‌های سطل‌های نامزد را محاسبه کرده و اثر انگشت در اندیس‌ها بررسی می‌کنیم. اگر با هر یک از مقادیر موجود در هر یک از سطل‌ها مطابقت داشت، یک نسخه از اثر انگشت از آن سطل حذف می‌شود.

الگوریتم ۲.۱۶: حذف مقدار از فیلتر فاخته

ورودی: مقدار $x \in D$

ورودی: فیلتر فاخته با اثر انگشت‌گذاری و تابع درهم‌ساز h

خروجی: True اگر مقدار حذف شده باشد و False در غیر این صورت

$f \leftarrow$ انگشت اثر (x)

$i \leftarrow h(x)$

$j \leftarrow i \oplus h(f)$

اگر $f \in CUCKOOFILTER[i]$ آنگاه

CUCKOOFILTER[i].drop(f)

برگرداندن True

در غیر این صورت اگر $f \in CUCKOOFILTER[j]$ آنگاه

CUCKOOFILTER[j].drop(f)

برگرداندن True

برگرداندن False

فیلتر فاخته دارای ویژگی‌هائی است که در ادامه شرح می‌دهیم.

الف- هنگامی که فیلتر فاخته نیاز به پشتیبانی از حذف دارد، باید چندین نسخه از مقدار را ذخیره کند یا برای هر مقدار ذخیره شده شمارنده‌هایی ترتیب دهد. با این حال، هر دو رویکرد حذف‌های احتمالی درست را القا می‌کنند، یکی - به دلیل ظرفیت محدود سطل

(نمی‌توانیم بیش از $۲b$ مقدار یکسان را در جدول ذخیره کنیم)، و دیگری به دلیل سرریز شمارنده‌ها، همانطور که در فیلتر بلوم شمارنده توضیح داده شد. با این حال، فیلتر فاخته غیرقابل حذف این مشکل را ندارد و از نظر فضا بسیار کارآمدتر است زیرا نیازی به به خاطر سپردن مقادیر یکسانی که چندین بار اضافه شده‌اند ندارد.

ب- مثبت کاذب امکان‌پذیر است. ممکن است مقادیر مختلف اثر انگشت یکسانی داشته باشند، اما در بیشتر موارد سطل‌های نامزد متفاوتی دارند، بنابراین هنوز هم می‌توان آنها را متمایز کرد. با این حال، زمانی که سطل‌های نامزد نیز برای آن مقادیر یکسان باشند، یک تصادم سخت رخ می‌دهد. به دلیل چنین رویدادهای بسیار نادری، فیلتر می‌تواند در پاسخ‌های مثبت کاذب قرار گیرد و احتمال آنها pr_{fp} برابر است با

$$pr_{fp} = ۱ - \left(۱ - \frac{۱}{۲^p}\right)^{۲b} \approx \frac{۲b}{۲^p}$$

معادله (۲.۱۲) نشان می‌دهد که با تعداد ثابت مقادیر مورد انتظار n، یک معاوضه بین احتمال مثبت کاذب pr_{fp} و اندازه سطل b وجود دارد که می‌تواند با طول اثر انگشت‌ها p جبران شود. به طور شهودی، اگر اثر انگشت‌ها به اندازه کافی طولانی باشند، درهم‌ساز فاخته با کلید جزئی تقریب خوبی از درهم‌ساز فاخته استاندارد است، اما اثر انگشت‌های طولانی‌تر بر فضای مورد نیاز تأثیر می‌گذارند. بنابراین، طول اثر انگشت توصیه شده p را می‌توان به صورت زیر تخمین زد:

$$p \geq \left\lceil \log \frac{۲b}{pr_{fp}} \right\rceil$$

و اگر بخواهیم در m سطل با اندازه b حداقل به اندازه تعداد مقادیر ورودی مقدار ذخیره کنیم، طول فیلتر با کران پایین زیر محدود می‌شود:

$$m \geq \left\lceil \frac{n}{b} \right\rceil$$

ج- منفی کاذب امکان‌پذیر نیست. مانند سایر موارد، اگر فیلتر فاخته عنصری را عضو نداند، قطعاً عضو مجموعه نیست:

$$pr_{fn} = ۰$$

فیلترهای فاخته اشغال فضای بالا را تضمین می‌کنند زیرا هنگام افزودن مقادیر جدید، تصمیمات قبلی در مورد قرارگیری مقادیر را اصلاح می‌کنند. با این حال، آنها حداکثر ظرفیتی دارند که با ضریب بار α بیان می‌شود. پس از رسیدن به حداکثر ضریب بار قابل اجرا، درج‌ها به طور فزاینده‌ای با احتمال شکست مواجه می‌شوند، بنابراین جدول درهم‌ساز باید گسترش یابد تا مقادیر بیشتری ذخیره کند.

میانگین تعداد بیت‌ها به ازای هر مقدار β را می‌توان به عنوان نسبت بین طول اثر انگشت‌ها و ضریب بار α تعریف کرد که با احتمال مثبت کاذب ثابت pr_{fp} می‌توان آن را به صورت زیر تخمین زد:

$$\beta \leq \frac{۱}{\alpha} \times \left\lceil \log \frac{۲b}{pr_{fp}} \right\rceil$$

چون طرح درهم‌ساز فاخته از دو تابع درهم‌ساز استفاده می‌کند، ضریب بار با سطل‌هایی با اندازه $b = ۱$ برابر با پنجاه درصد است زیرا جدول درهم‌ساز مستقیماً نگاشت می‌شود. با این حال، افزایش اندازه سطل امکان بهبود اشغال جدول را فراهم می‌کند، به عنوان مثال، برای $b = ۲$ و $b = ٤$ ضریب بار به ترتیب ۸۴ درصد و ۹۵ درصد است.

مطالعه تجربی نشان داد که سطل‌هایی با اندازه $b \in \{۱,۲,۳,۴\}$ برای موارد عملاً مهم کافی هستند.

مثال ۲.۱۴: حافظه مورد نیاز

می‌خواهیم یک میلیارد مقدار را با فیلتر فاخته مدیریت کنیم، احتمال رویدادهای مثبت کاذب را حدود دو درصد و اشغال جدول را ۸۴ درصد نگه داریم. برای پشتیبانی از چنین ضریب باری، اندازه سطل‌ها را $b = ۲$ انتخاب می‌کنیم، به این معنی که طول فیلتر $m = ۲^{۲۹}$ است، با توجه به (۲.۱۴).

همانطور که در (۲.۱۳) بیان شد، حداقل طول اثر انگشت است

$$p = \left\lceil \log \frac{۴}{۰.۰۲} \right\rceil = ۸$$

بنابراین، طول مورد نیاز اثر انگشت ۸ بیت است و اندازه کل فیلتر فاخته $۲^{۲۹} \times ۸ \times ۲$ بیت است که تقریباً ۱.۰۷ گیگابایت حافظه است.

برای مقایسه نیازهای فضایی با سایر فیلترهای مطالعه شده، می‌توانیم از $b = ۱$ استفاده کنیم که به ۵۰ درصد اشغال جدول دست می‌یابد و نیاز به فیلتری به طول $m = ۲^{۳۰}$ دارد. با توجه به (۲.۱۳)، باید از اثر انگشت‌های ۹ بیتی استفاده کنیم که منجر به حدود ۰.۹۴ گیگابایت حافظه می‌شود.

در واقع، فیلترهای فاخته از رویکردی مشابه فیلتر بلوم شمارنده استفاده می‌کنند، اما کارایی فضایی بهتری و پیاده‌سازی بسیار ساده‌تری دارند. برای برنامه‌هایی که مقادیر زیادی را ذخیره می‌کنند و میزان مثبت کاذب نسبتاً پایینی (کمتر از ۳ درصد) را هدف قرار می‌دهند، فیلترهای فاخته فضای کمتری نسبت به حتی فیلترهای بلوم بهینه‌سازی شده ارائه می‌دهند.

با این حال، زمانی که فیلتر فاخته در حداکثر ظرفیت خود است، جدول درهم‌ساز زیرین باید گسترش یابد، تا آن زمان امکان افزودن مقادیر جدید وجود ندارد. در مقابل، با فیلتر بلوم هنوز می‌توان به قیمت افزایش نرخ مثبت کاذب به درج مقادیر جدید ادامه داد.