**موجودی عجیب! به نام Bloom Filter در برنامه نویسی**

بلوم فیلتر و کاربرد آن در برنامه نویسی

خُب این بلوم فیلتر چیه و به چه دردی میخوره را من هم نمی‌دونستم تا اینکه توی مقاله‌ای برای کاهش مصرف حافظه و کاهش افزونگی داده‌های تبادل شده بین نودهای شبکه‌های تحمیل پذیر تاخیر یا DTN بهش برخوردم که البته الان DTN موضوع مورد بحثم نیست و بعدا در یک نوشته جدا بهش می‌پردازم.

*فرض کنید می‌خواهیم****وجود یک عنصری****را در مجموعه جستجو کنیم، اگر ساختار داده (Data Structure) به شما جواب داد که این عضو در مجموعه وجود دارد احتمال دارد که وجود نداشته باشد.****اما اگر بگوید وجود ندارد****، قطعاً درست است. بر این اساس****خطای مثبت****داریم ولی****خطای منفی****امکان پذیر نمی‌باشد.*

احتمالا شما هم مثل من گیج شدید :)

فرض کنید ما یک برگه کاغذ داریم که فقط و فقط مشخصات داده‌هایی که وارد میشن را می‌نویسیم دقت کنید ما در مورد داده‌های که خارج میشن چیزی نمی‌نویسیم. به این معنی که ممکنه یک داده‌ای وارد شده باشه و بعد هم خارج شده باشه و ما فقط زمان ورود اسمش را ثبت کردیم. بزارید با یک مثال پیش بریم. فرض کنید متغیرهای X,U,A,H وارد شدند. حالا اگر یک نفر از ما سوال کنه که داده X داخل لیست هست یا نه؟ ما چه جوابی می‌تونیم بدیم؟ همونطور که شما هم دارید حدس می‌زنید، درسته ما می‌تونیم بگیم X در لیست وجود داره اما این را هم در نظر داشته باشید که یک احتمال هم هست که متغیر X توی لیست نباشه و خارج شده باشه و از اونجایی که ما فقط و فقط اسامی ورودی ها را یادداشت می‌کنیم این توضیح در مورد خطای مثبت و خطای منفی درست هست. توجه داشته باشید ما نمی‌رویم داخل مجموعه‌ای که عنصرها داخلش ذخیره شدند جست و جو انجام بدیم و ببینیم آیا هنوز عنصر وارد شده هست یا نه، ما فقط یک لیست ثبت نامی یا همون برگه کاغذ را داریم که فقط و فقط لیست ورودی ها را داخلش می‌نویسیم. مثلا ما نمی‌رویم داخل دانشگاه تک تک اتاق اساتید را بررسی کنیم و دنبال آقای X بگردیم. بجای این کار فقط لیست اسامی وارد شده درب ورودی را چک می‌کنیم.

حالا بیاید دنبال Y بگردیم و خب نتیجه جست و جو به ما میگه که Y در لیست نیست. در این حالت ما **قطعا** می‌تونیم بگیم که Y وجود نداره چرا؟ چون اصلا وارد نشده هنوز تا اسمش توی کاغذ ما ثبت بشه.

یک سوال مهمی که پیش میاد این هست که این ساز و کار به چه دردی می‌خوره و چه کاربردی داره؟ کجاها می‌تونه باعث کاهش افزونگی و افزایش سرعت بشه؟

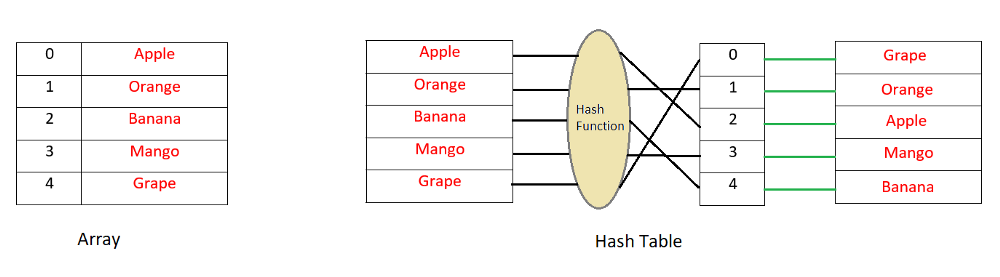
با این مقدمه بریم سراغ معرفی **ساختار داده‌های احتمالاتی** و **بلوم فیلتر** و کاربردش در **برنامه نویسی**

**تعریف Bloom filter**

احتمالا می‌دونید که **[جدول درهم سازی](https://fa.wikipedia.org/wiki/%D8%AC%D8%AF%D9%88%D9%84_%D8%AF%D8%B1%D9%87%D9%85%E2%80%8C%D8%B3%D8%A7%D8%B2%DB%8C" \t "_blank)**(hash table) چطور کار می کنه؟ اگر نمی‌دونید پیشنهاد می‌کنم یک سری به [ویکی](https://fa.wikipedia.org/wiki/%D8%AC%D8%AF%D9%88%D9%84_%D8%AF%D8%B1%D9%87%D9%85%E2%80%8C%D8%B3%D8%A7%D8%B2%DB%8C" \t "_blank) بزنید.

در علوم رایانه، جدول درهم‌سازی یا جدول هش نوعی ساختمان داده است که مقدارهایی[۱] که باید ذخیره شوند را به وسیله تابع هش (درهم سازی) با کلیدهای ویژه‌ای مرتبط می‌سازد. عملیات اولیه‌ای که این جدول تسهیل می‌کند عمل مراجعه است. به این معنی که کاربر می‌تواند با سرعتی کارآمد داده مورد نظرش را در آن بیابد. در جدول‌های هش همچنین افزودن داده‌های جدید در زمان کم امکان‌پذیر است. زمان لازم برای جستجو و افزودن وابسته به نوع جدول و میزان داده‌ها هستند. این زمان می‌تواند با انتخاب جدول مناسب به مرتبه زمانی O(1) برسد.

وقتی شما یک داده جدید در یک آرایه یا لیست ساده وارد می کنید، index همون مکانی که داده در آن قرار داده می شه از روی داده ای که داره تشخیص داده نمی‌شه. این به این معنی هست که ارتباط مستقیمی بین index و مقدار ذخیره شده در اون خونه از آرایه وجود ندارد.به همین دلیل اگر شما نیاز داشته باشید که یک مقدار را در آرایه جست و جو کنید، مجبور خواهید شد که همه Index ها را جست و جو کنید تا داده مورد نظرتون را پیدا کنید. اما در جدول درهم‌ساز شما key یا index را توسط هش کردن مقدار یا value تشخیص می دید. بعد این مقدار را در این index از لیست قرار می دید. به این معنی که key از روی مقدار value تشخیص داده می‌شه و هر زمانی شما نیاز به بررسی اینکه آیا value موجود هست یا نه داشته باشید شما فقط مقدار value را hash‌ می‌کنید و بعد بر اساس key جست وجو می‌کنید که توی تصویر زیر نشون داده شده. این خیلی سریعتر هست و مرتبه الگوریتم نیز O(1) است که خب خیلی سریع‌تر از O(n) است.

نحوه عملکرد جدول درهم سازی یا hash table

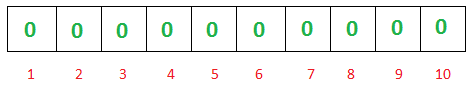
خب تا اینجا مقدمات اصلی یا بهتره بگم پیش نیازهایی در مورد بلوم فیلتر را با هم بررسی کردیم و دیگه رسیدیم به قسمت های ساده تر و اصلی، بزارید با یک مثال کمی ملموس‌تر توی دنیای برنامه نویسی جلو بریم. بیاید در نظر بگیریم که یک لیست بزرگی از پسوردهای ضعیف وجود دارد و آن‌ها روی برخی از سرورهای remote ذخیره شده‌اند. امکان لود همه پسوردها بصورت فوری در حافظه memory/RAM به خاطر فضای کم وجود ندارد. هر زمان یک کاربر پسورد خودش را وارد می‌کند شما می‌خواهید بررسی کنید که اگر پسورد ضعیف بود به کاربر یک هشدار بدید. چه کار می‌تونید بکنید؟ در نظر بگیرید که شما لیست پسوردهای ضعیف را دارید، شما می تونید این‌ها را در یک **جدول درهم ساز** یا چیزی مشابه ذخیره کنید و هر زمان که می‌خواهید بررسی کنید، به جای جستجوی داده اصلی شما می توانید چک کنید که آیا هیچ گزینه متناظری وجود دارد یا نه. این تناظر یا تشابه ممکن است سریع باشد اما هزینه جست و جوی آن روی دیسک یا روی شبکه در یک سرور ریموت ممکنه زیاد یا باعث کندی باشه.

فراموش نکنید که شما این کار را برای هر پسوردی که توسط کاربر وارد میشه باید انجام بدید. چطور میشه هزینه این کار را کاهش داد؟

**بلوم فیلتر** میتونه به ما کمک کنه. اما چطوری؟ بریم ببینیم بلوم فیلتر چطور کار می‌کنه.

طبق تعریف **بلوم فیلتر** می‌تونه یک مقدار را در حالت‌های “**احتمال وجود آن در مجموعه است**” یا “**قطعا در آن مجموعه نیست**” بررسی کند. تفاوت ظریف بین **احتمالا** و **قطعا نه** در اینجا بسیار مهم است.

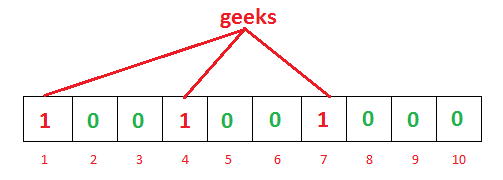
به خاطر این جمله “احتمالا در مجموعه است“ می‌باشد که چرا این ساختار داده‌ها **احتمالاتی** نامیده می‌شوند. و با تعریف هایی که شده بدین معنی است که **مثبت کاذب** امکان پذیر هست اما **منفی کاذب** غیر ممکن است. عجله نکنید، در ادامه با مثال عملی می‌بینیم که دقیقا اینا یعنی چی. بلوم فیلتر اساسا شامل یک **بردار بیتی** یا لیست بیتی(**یک لیست شامل مقدار بیت فقط 0 یا 1**) به **طول m**، با **مقدار اولیه 0**مانند شکل زیر است:



مقداردهی لیست بیتی بلوم فیلتر

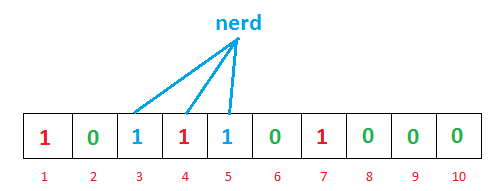
برای اضافه کردن یک آیتم به بلوم فیلتر، ما اول آیتم مون را به**k تابع هش** مختلف می دهیم هر تابع هش یک عدد به ما می‌دهد که ما متناظر با هر عدد توی آرایه‌مون مقدار اون خونه را برابر 1 قرار می‌دهیم. همونطور که می‌تونید ببینید در جدول هش ما از یک تابع هش تکی استفاده کردیم و به عنوان نتیجه فقط یک index تکی به عنوان خروجی داشتیم.

اما در این مثال از بلوم فیلتر ما از تابع هش چندگانه (k تعداد توابع هش) استفاده می‌کنیم که خروجی چندگانه نیز خواهیم داشت.



خروجی تابع هش چندگانه برای کلمه geeks

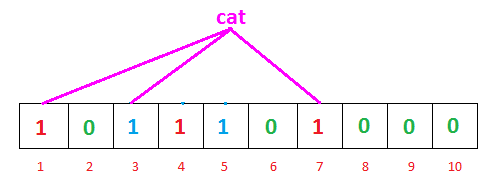
همونطور که در تصویر بالا مشخص هست برای ورودی آیتم geeks ما سه تابع هش داریم که هر تابع یک مقدار به ما بر می‌گردونه که در این مثال مقدار 1 ,4,7 هست یعنی خونه شماره 1 , 4 و 7 را از صفر به 1 تغییر میدهیم که ما آن ها را مشخص کردیم.



خروجی تابع هش چندگانه برای کلمه nerd

به همین ترتیب برای ورودی دیگری به نام nerd توابع هش به ما ۳ خروجی 3,4,5 را می‌دهد.

شاید متوجه شده باشید که index شماره 4 یا همون خونه شماره 4 قبلا با کلمه geeks علامت گذاری شده(از صفر به 1 تغییر داده شده). خب تا اینجا با بردار بیتی خودمان را با دو وردی پر کرده ایم. و حالا می تونیم بررسی کنیم که آیا مقدار مورد نظر ما داخل آن وجود دارد یا نه. اما چطوری؟ خیلی ساده هست. ما یک جدول هش ایجاد کردیم. ورودی مورد نظرمون که می‌خواهیم جست و جو کنیم را با سه تا تابع هش می‌کنیم و نتیجه سه تا تابع که سه تا عدد است را بررسی می کنیم که کدام index‌ها یا ساده‌تر کدوم مکان‌های آرایه میشه. مثلا اگر بخواهیم ببینیم کلمه nerd وجود دارد یا نه کافیه ابتدا این کلمه را بدیم به سه تابع هش و خروجی که سه تا عدد است را داخل آرایه تک بعدی بررسی کنیم. که خب هر سه تا خونه مقدارش برابر 1 است یعنی این عنصر احتمالا وجود دارد.



جستجوی کلمه cat در بلوم فیلتر

در ادامه بزارید جست و جو برای کلمه cat را انجام دهیم که خروجی سه تابع هش این کلمه برابر 1 ,3,7 هست که می‌تونیم ببینیم که مقدار هر سه تا خونه 1,3,7 برابر 1 هست که این به این معنی است که ما می‌توانیم بگوییم “احتمالا کمله cat در لیست وارد شده است”. **اما وارد نشده است**!!

پس این خطا از کجا است؟ در واقع هیچ خطایی رخ نداده. به این اتفاق میگن خطای مثبت کاذب که در ادامه نحوه جلوگیری از این اتفاق را هم توضیح می‌دهم. بلوم فیلتر به ما می گوید که این نشان می دهد که شاید کلمه cat قبلا وارد شده بوده است زیرا همه خانه‌های index مربوط به خروجی سه تابع هش برای کلمه cat برابر 1 است(حتی با داده های مختلف دیگه). خب این کجاش مفیده و بدرد می‌خوره؟

بیاید در نظر بگیریم اگر خروجی توابع هش برای کلمه cat به جای 1,3,7برابر 1,6,7 می‌بود، در این صورت چی می‌شد؟ ما می تونستیم ببینیم که **Index مربوط به 6 برابر 0**است. که به این معنی هست که با هیچ ورودی، قبلا این خونه علامت‌گذاری(تغییر از صفر به 1) نشده. که به این معنی است که کلمه cat **هیچ وقت** وارد نشده است. این **نحوه کار بلوم فیلتر** است که می تواند به ما بگوید “**دقیقا**”داده در لیست وجود ندارد. خب به طور خلاصه:

* اگر ما برای یک مقدار جست و جو کردیم و نتیجه هر هش‌ای را پیدا کردیم که مقدار آن **برابر 0**بود ما می‌توانیم **با اطمینان** بگوییم که در لیست **وجود ندارد**.
* اگر همه indexهای خروجی توابع هش برابر 1 باشد آنگاه **ممکن است** مقدار جست و جو شده وجود داشته باشد.

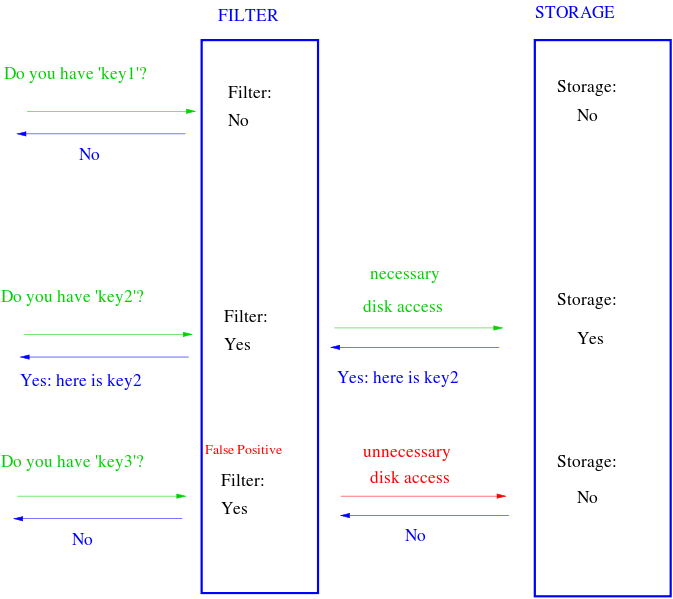
در واقع در برخی شرایط ما می‌توانیم به جای جستجوی موجود بودن گزاره را برعکس کنیم. و دنبال نبودن یا شامل نشدن بگردیم و خیلی سریع‌تر به جواب برسیم.

خب بزارید برگردیم به مثال پسورد که ابتدا در موردش صحبت کردیم. اگر ما بررسی پسوردهای ضعیف را با این نوع از **بلوم فیلتر** پیاده سازی کنیم، ابتدا، ما بلوم فیلترمان را با یک لیست از پسوردهای ضعیف که به ما یک بردار بیتی می‌دهد که برخی از indexهای آن با 1مقدار دهی شده و مابقی با 0 مقدار دهی شده است، ایجاد می‌کنیم. از آنجایی که اندازه بلوم فیلتر نمی تواند خیلی بزرگ باشد و یک اندازه ثابت دارد، این می تواند به راحتی در حافظه ذخیره شود حتی در صورت نیاز می‌تواند در سمت client ذخیره شود.

به همین دلیل است که بلوم فیلتر بسیار space-efficient است. در حالی که یک جدول هش نیاز به داشتن اندازه دلخواه بر اساس داده‌های ورودی دارد، بلوم فیلترها می توانند با **اندازه ثابت** به خوبی کار کنند و این یعنی عملیات جستجو خیلی سریع‌تر و ساده‌تر خواهد بود.

بنابراین هر بار که کاربر پسورد خود را وارد می کند ما آن را به توابع هش می‌دهیم و خروجی توابع هش که مکان خانه‌های آن در بردار بیتی است را بررسی می‌کنیم که وجود نداشته باشد. اگر پسورد به اندازه کافی قوی باشد بلوم فیلتر به ما نشان می دهد که پسورد **دقیقا در لیست پسورد های ضعیف نیست** و نیازی نیست که ما کوئری سمت سرور بزنیم. اما اگر پسورد ضعیف بود و به ما جواب مثبت(ممکن است مثبت کاذب باشد) داده شود، ما یک درخواست سمت سرور برای بررسی لیست اصلی می کنیم. همانطور که می‌بینید اغلب اوقات ما حتی نیاز به درخواست از سرور و یا از دیسک برای چک کردن لیست نداریم که این باعث بهبود قابل توجهی در سرعت نرم افزار می‌شود چرا که صرفا یک بردار بیتی را بررسی می‌کنیم و سمت دیتابیس کوئری نمی‌زنیم. در صورتی که بخواهیم بردار بیتی را در سمت سرویس گیرنده یا client ذخیره کنیم، هنوز هم می توانیم آن را در حافظه سرور بارگزاری کنیم و حداقل برخی از زمان جست و جوی دیسک را صرفه جویی کنیم. به بیان ساده‌تر به جای مراجعه به لیست بلندی از پسوردها و جست و جو داخلشون که از مرتبه n یا دیگر مرتبه ها است ما از روی پسوردها یک بردار بیتی درست می‌کنیم و برای بررسی کلمه مورد نظر را هش می‌کنیم مثلا ۳ تا خروجی میده فقط ۳تا خونه بردار بیتی را بررسی می کنیم که سرعت کار ما را خیلی بالا می‌بره.

همچنین در نظر بگیرید که اگر نرخ خطای مثبت بلوم فیلتر شما 1% باشد به این معنی است که در میان هزینه های مراجعه به سرور یا دیسک، تنها 1% از کوئری‌های ما با خطا مواجه خواهد شد و ۹۹درصد دیگر سمت سرور نخواهند رفت. بد نیست نه؟

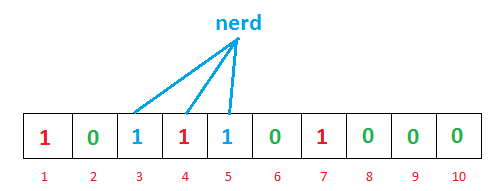
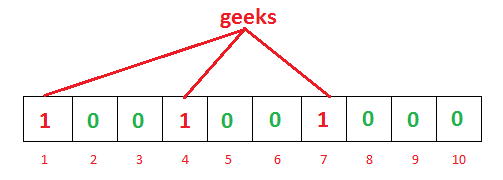


**عملیات‌های بلوم فیلتر**

یک بلوم فیلتر ساده از دو عملیات test (یا بررسی) و add (یا اضافه کردن) پشتیبانی می‌کند. Test برای چک کردن اینکه آیا یک عنصر در مجموعه وجود دارد یا نه و عملیات add هم برای اضافه کردن یک عنصر به مجموعه.

به نظرتون امکان حذف کردن یک ایتم از بلوم فیلتر وجود دارد؟ قبل از ادامه دادن یکم روی جواب تون فکر کنید.

قبل از جواب دادن به سوال بالا بیاید یک مثال بصورت عملی بررسی کنیم. فرض کنید در بردار بیتی دو کلمه geeks و nerd را اضافه کردیم:



حالا ما میخواهیم عنصر geeks را از بلوم فیلتر حذف کنیم. بنابراین اگر ما 1,4,7 را از بردار بیتی حذف کنیم که قبلا توسط geeks علامت گذاری شده بودند، و ما مقدارشونا به 0 تغییر بدیم چه اتفاقی می افتد؟

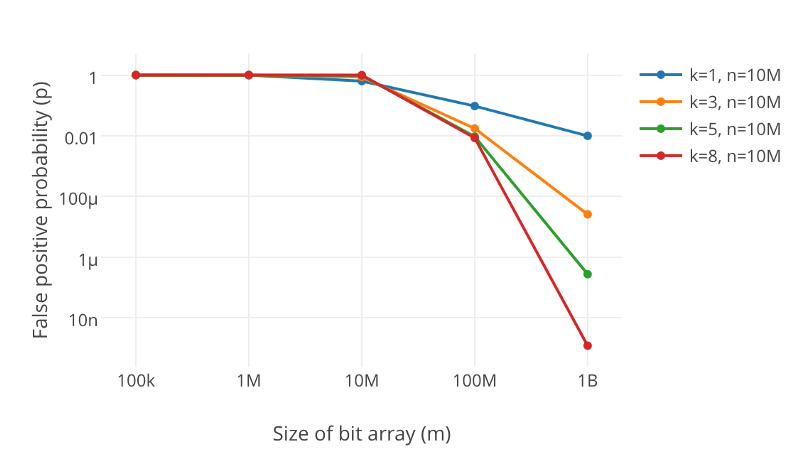
به سادگی می‌تونید ببینید که اگر ما در ادامه کلمه nerd را جست و جو کنیم همانطور که می‌بینید Index شماره 4 برابر 0 هست و این به ما می‌گوید که دقیقا nerd در لیست وجود ندارد در حالی که اشتباه است. این بدین معنی است که **امکان حذف در بلوم فیلتر وجود ندارد**.

پس راه حل چیه؟ در واقع توی بلوم فیلتر ساده چنین امکان وجود نداره. اما اگر واقعا به چنین چیزی نیاز دارید می تونید از یک نسخه تغییر یافته بلوم فیلتر به نام Counting bloom filter استفاده کنید. ایده ساده است. به جای مرتب سازی یک بیت تکی از مقدارها، ما یک مقدار integer را ذخیره می کنیم و بردار بیتی ما بردار integer خواهد بود. قابلیت حذف کردن باعث افزایش اندازه و هزینه ما خواهد شد. در این حالت به جای علامت گذاری یک مقدار بیت با 1 هنگام ورود داده، ما مقدار integer را به مقدار 1 افزایش می دهیم و اگر داده جدیدی وارد بلوم فیلتر کردیم یک واحد دیگه افزایش می‌دهیم مثلا مقدار خانه اگر 1 بود با افزایش مجدد به 2 تغییر داده می‌شود و در هنگام حذف کلمه یک واحد از خانه‌هایی خروجی توابع هش کم می‌کنیم. برای بررسی اینکه یک عنصر موجود هست یا نه کافی هست که عنصر را هش کنیم و خانه هایی را که بررسی میکنیم مقدارشون بیشتر از 0 باشه. که قابلیت حذف را ما فعلا نیاز نداریم و بیشتر از این هم بهش نمی‌پردازیم.

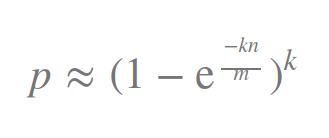
**اندازه Bloom filter و تعداد توابع Hash**

شاید به این فکر کرده باشید که اگر اندازه بلوم فیلتر کوچک در نظر گرفته بشه ممکنه سریع همه خونه های آن برابر 1 بشه و بعد از این هر کلمه‌ای چه در لیست باشه چه نباشه را بررسی کنیم جواب مثبت برگردانده بشه. بنابراین اندازه بلوم فیلتر خیلی مهم است. در اینجا اگر اندازه بلوم فیلتر بزرگ انتخاب شود ما جواب های مثبت کاذب کمتری داریم و اگر اندازه کوچک باشد ما جواب های مثبت کاذب بیشتری خواهیم داشت. اینجا یک موضوع جدید خواهیم داشت با عنوان **نرخ خطای مثبت کاذب**. پارامتر مهم دیگر این است که چه تعداد تابع هش نیاز داریم؟

اینجا است که تحقیقات دانشگاهی می‌تواند به کمک ما بیاد(این موضوع را برای این اشاره کردم که بعضی از دوستان فکر می‌کنند دانشگاه در مقطع تحصیلات تکمیلی به چه دردی میخوره؟ دقیقا به درد کارهای تحقیقاتی می‌خوره که اینجا یک نمونه‌ش را اشاره می‌کنم).

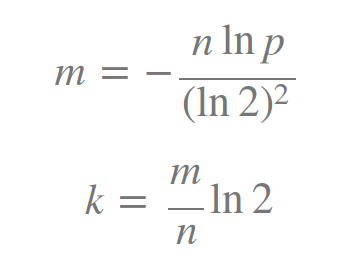
طبق [مقاله ارائه شده](https://www.eecs.harvard.edu/~michaelm/postscripts/rsa2008.pdf" \t "_blank) که نتیجه آن در شکل زیر نشان داده شده می‌توانید ببینید که اندازه بلوم فیلتر و تعداد توابع هش به صورت مناسب چقدر هست:

در نمودار بالا با افزایش تعداد تابع هش یعنی k، نرخ خطا که با p نشان داده شده کاهش پیدا می‌کند. در واقع ما می‌توانیم نرخ خطای کاذب مثبت P را بر اساس اندازه بلوم فیلتر که با m و تعداد تابع هش که با k نشان می‌دهیم، اگر تعداد عنصر ورودی ما(مثلا تعداد پسوردها) برابر n باشد، می‌توانیم بصورت زیر محاسبه کنیم:



محاسبه احتمال خطای کاذب برای بلوم فیلتر

اگر به ریاضیات علاقه ای ندارید نگران نباشید :) در واقع ما فقط نیاز داریم که مقدار m و k را مشخص کنیم. اگر نرخ خطای مثبت کاذب ما برابر p باشد و تعداد داده‌های ورودی برابر n باشد ما می تونیم از فرمول های زیر برای محاسبه پارامترهامون استفاده کنیم:



محاسبه اندازه بلوم فیلتر m و تعداد توابع هش مورد نیاز k

نکته دیگه که باید توجه کنید این هست که هدف استفاده از بلوم فیلتر جست و جوی سریعتر هست، ما نمی توانیم از توابع هش با سرعت پایین استفاده کنیم. بنابراین توابع هش رمزنگاری **Sha-1** و **MD5** انتخاب خوبی برای بلوم فیلترها نیستند. توابع هش سریعتری پیاده سازی شدن که می‌توانید ازشون استفاده کنید. مثلا [murmur fnv HashMix](https://en.wikipedia.org/wiki/MurmurHash" \t "_blank) انتخاب های بهتری هستند.

**کاربردها**

بلوم فیلتر برای بررسی عضو داخل یک مجموعه است. مثال ساده بلوم فیلتر برای کاهش هزینه مراجعه به دیسک یا شبکه برای کلیدهای ناموجود است. همونطور که ما می‌توانیم ببینیم بلوم فیلترها می توانند یک کلید را در O(k) جست و جو کنند که k تعداد تابع هش است. این بسیار سریعتر هست که کلید ناموجود را جست و جو کنیم. اگر عنصر در بلوم فیلتر نباشد پس ما نیازی به جست و جوی پر هزینه نداریم. از طرف دیگه اگر عنصر در بلوم فیلتر موجود باشد ما جست و جو را انجام می‌دهیم و می‌توانیم انتظار داشته باشیم که در برخی موارد ما نرخ خطای مثبت کاذب(false positive rate) داشته باشیم. و عنصر موجود نباشد. در ادامه برخی مثال‌های استفاده از بلوم فیلتر را معرفی می‌کنیم.

من در این قسمت در مورد یکی از کاربردهایی که برای پایان نامه ام درگیرش بودم کمی توضیح می‌دهم که چطور استفاده از بلوم فیلتر توی کاهش سربار اطلاعات تبادل شده در شبکه‌های تحمل پذیر تاخیر میتونه مفید باشه.

خیلی خلاصه شبکه های تحمل پذیر تاخیر یا DTN شبکه هایی هستند که بر خلاف شبکه های سنتی که ارتباط end-to-end اگر با تاخیر مواجه بشه ارتباط را خاتمه میده می تونن تاخیر را تا حدود زیادی تحمل کنند. در صورت نیاز به مطالعه بیشتر [اینجا](https://en.wikipedia.org/wiki/Delay-tolerant_networking) را می‌توانید مطالعه کنید. در این شبکه ها دو تا گره(node) که همون دستگاه‌ها هستند حالا می‌تواند یک ماشین مجهز به wifi توی بزرگراه هوشمند باشه یا اینکه یک ماهواره در مدار زمین باشه وقتی با همدیگه مواجه می‌شوند باید بسته‌هایی که در بافر خودشون دارند را با یکدیگر مبادله کنند. یک راه ساده انتقال کل بافر هر نود برای نود دیگه است که خب سربار خیلی زیادی داره و از طرف دیگه ممکنه یک نود برخی از پکت‌ها را قبلا توسط بقیه نودها دریافت کرده باشه مجددا دریافت کنه و باعث سربار اضافی بشه. برای کاهش سربار و جلوگیری از ارسال تکراری پکت ها استفاده از بلوم فیلتر یک راهکار خوبی است. به این صورت که ابتدا هر نود یک بردار بیتی(بلوم فیلتر) از بسته هایی که دارد ایجاد می‌کند که حجم بسیار پایینی دارد و آن را برای نود دیگه ارسال می‌کند. و نود قبل از ارسال بسته‌ها چک ‌می‌کند که آیا داخل بلوم فیلتر دریافتی وجود دارند یا نه. اگر بلوم فیلتر جواب منفی داد یعنی بسته‌ها در نود دیگر وجود ندارند و باید ارسال بشوند و اما اگر بلوم فیلتر اعلام کنه که بسته موجود است از ارسال آن جلوگیری میشه. که این می‌تواند باعث کاهش سربار تبادلات شود.

**مثال های استفاده از بلوم فیلتر:**

* برای هشدار به کاربران در مورد ضعیف بودن پسورد که در بالا توضیح داده شد.
* برای جلوگیری از دسترسی کاربران به سایت های مخرب
* برای بررسی اینکه چه کاربرانی قبلا از سایت شما بازدید کردند بر اساس IP هاشون.
* برای بررسی اینکه چه تعداد از کاربران محتوای سایت شما را خوانده اند