যদি তোমার একটি নিজস্ব ওয়েবসাইট থাকে তাহলে অবশ্যই সেখানে একাধিক ইউজার থাকবে। আর প্রতিটি ইউজারের একটি নিজস্ব অ্যাকাউন্ট থাকবে যেখানে তারা নিজেদের ইউজারনেম এবং পাসওয়ার্ড দিয়ে লগইন করতে পারবে। এখন নতুন কোনো ইউজার যদি অ্যাকাউন্ট খুলতে চায় তখন তোমাকে খেয়াল রাখতে হবে যে, সে যেই ইউজারনেমটি ব্যবহার করছে সেই একই ইউজারনেম ব্যবহার করে ইতোমধ্যেই অন্য কেও অ্যাকাউন্ট তৈরি করেছে কিনা।

Username mynamewastaken

username has already been taken

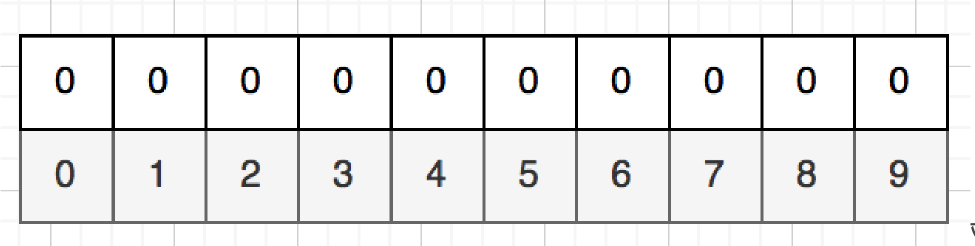
Your URL: http://twitter.com/mynamewastaken

ধরা যাক, সবগুলো ইউজারনেম তুমি আগেই একটি ডাটাবেসে সেভ করে রেখেছো। প্রতিবার যখন নতুন ইউজার রেজিস্টার করার চেষ্টা করে তখন কিভাবে তুমি চেক করবে যে ইউজারনেমটি ডুপ্লিকেট নাকি? সহজ উপায় হলো প্রতিবার ইউজারনেমটিকে সার্ভারে পাঠিয়ে ডাটাবেসে চেক করা।

এখন মনে করো তোমার ওয়েবসাইট খুবই বিখ্যাত, ইউজার আছে ১০ লাখ এবং প্রতি মিনিটেই কয়েক হাজার নতুন ইউজার নতুন অ্যাকাউন্ট তৈরি×। সেক্ষেত্রে ডুপ্লিকেট চেক করার জন্য সার্ভারে অনেক রিকুয়েস্ট পাঠাতে হবে, ডাটাবেসও এক্সেস করতে হবে অনেকবার। এক্ষেত্রে তুমি একটা ক্যাশ ব্যবহার করে ডাটাবেস এক্সেস কমিয়ে আনতে পারো কিন্তু সবথেকে ভালো হয় যদি সার্ভারে রিকুয়েস্ট না পাঠিয়েই আমরা ডুপ্লিকেট চেক করতে পারি। একটা উপায় হলো ১০লাখ ইউজারের লিস্ট ক্লায়েন্টের ব্রাউজারে পাঠিয়ে দেয়া এবং সেখানে ডুপ্লিকেশন চেক করা কিন্তু তাহলে প্রচুর ডাটা ব্রাউজারে পাঠাতে হবে, এবং ইউজারনেমের তালিকাও ফাঁস হয়ে যাবে সবার কাছে।

এই ধরণের সমস্যা সমাধানের জন্য ব্যবহার করা হয় ব্লুম ফিল্টার নামক একটা প্রোবাবিলিস্টিক ডাটা স্ট্রাকচার। এটা খুবই সিম্পল একটা ডাটা স্ট্রাকচার, তোমার যদি হ্যাশিং নিয়ে ধারণা থাকে তাহলে বুঝতে কোনো সমস্যা হবে না।

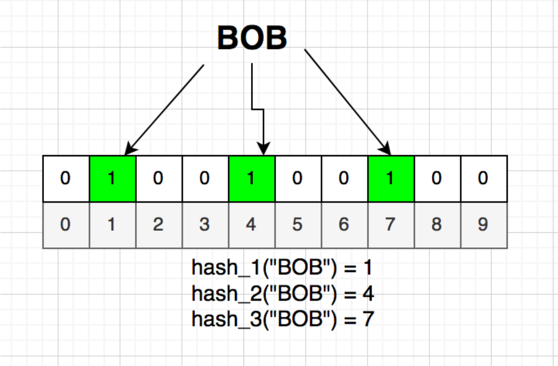
ব্লুম ফিল্টারের বৈশিষ্ট হলো তুমি যদি এখানে একটা ইউজারনেম ইনপুট দাও তাহলে সে দুইরকম আউটপুট দিবে, একটি হলো, “আমি নিশ্চিত ইউজার নেমটি ডাটাবেজে নাই”; আরেকটি হলো, “আমি *প্রায়* নিশ্চিত যে ইউজারনেমটি ডাটাবেসে আছে”। এখানে লক্ষ্য করার বিষয় হলো ব্লুম ফিল্টার যদি বলে যে ইউজার নেমটি ডাটাবেসে নাই তাহলে ১০০% নিশ্চিত যে নামটি নেই; কিন্তু যদি বলে ডাটাবেসে আছে তাহলে সেটা থাকতেও পারে, নাও থাকতে পারে এবং নিশ্চিত হতে হলে তোমাকে ডাটাবেসে খুজে দেখতে হবে।
ব্লুম ফিল্টার হলো m বিটের একটি ভেক্টর। একদম শুরুতে সবগুলো বিটের মান শূন্য।



0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

আমাদের এই উদাহরণে ধরে নিবো $m=9m=9$। এখন আমাদেরকে kk টা হ্যাশ ফাংশন ডিফাইন করতে হবে। প্রতিটা হ্যাশ ফাংশনের কাজ হলো একটি ইউজারনেম ইনপুট হিসাবে নিয়ে সেটাকে mm সাইজের বিট ভেক্টরের একটি ঘরে অ্যাসাইন করা। উদাহরণ হিসাবে ধরে নিলাম $k=3k=3$।

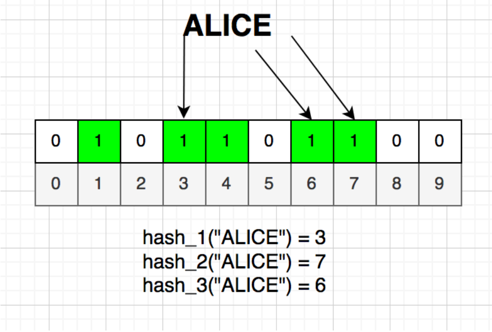
এখন মনে করো প্রথম ইউজারের নাম হলো “BOB”। “BOB” কে 3 টা হ্যাশ ফাংশন দিয়ে হ্যাশ করে আমরা ধরো পেয়েছি 1, 4 এবং 7। আমরা ব্লুম ফিল্টারে সেই বিটগুলোকে অন করে দিবো।



0	1	0	0	1	0	0	1	0	0
0	1	2	3	4	5	6	7	8	9

hash_1("BOB") = 1
hash_2("BOB") = 4
hash_3("BOB") = 7

২য় ইউজারের নাম ALICE এবং ALICE কে হ্যাশ করে পেয়েছি 3, 7 এবং 6। আমরা বিটগুলোকে অন করে দেই।

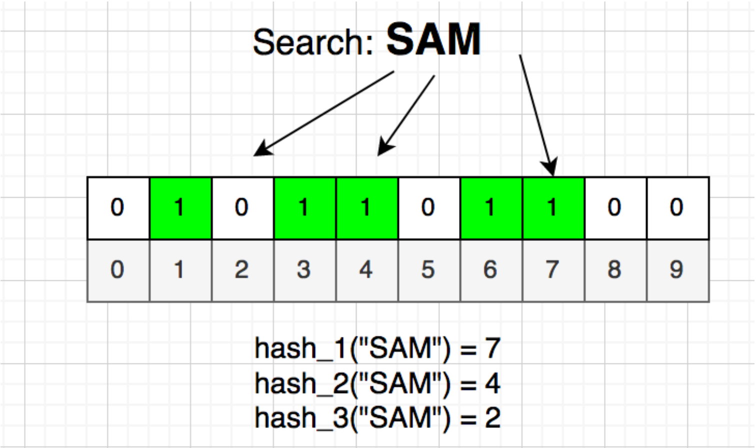


0	1	0	1	1	0	1	1	0	0
0	1	2	3	4	5	6	7	8	9

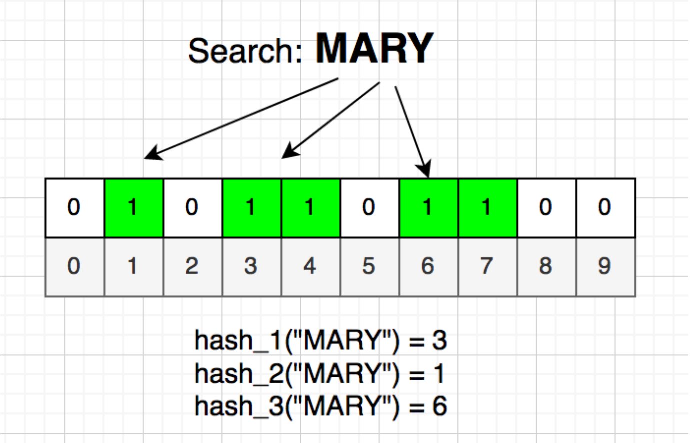
hash_1("ALICE") = 3
hash_2("ALICE") = 7
hash_3("ALICE") = 6

এখন আমাদের ব্লুম ফিল্টারে ALICE এবং BOB নামদুটি সেভ করা আছে।

এখন নতুন একটা শব্দ ব্লুম ফিল্টারে আছে নাকি নেই সেটা বুঝতে হলে তোমাকে আগে সেই নামটাকে kk বার হ্যাশ করতে হবে এবং kk টি পজিশনে গিয়ে দেখতে হবে বিটটি অন আছে নাকি। যদি তুমি অন্তত একটি বিট অফ পাও তার মানে শব্দটি নিশ্চিতভাবেই ব্লুম ফিল্টারে নেই।



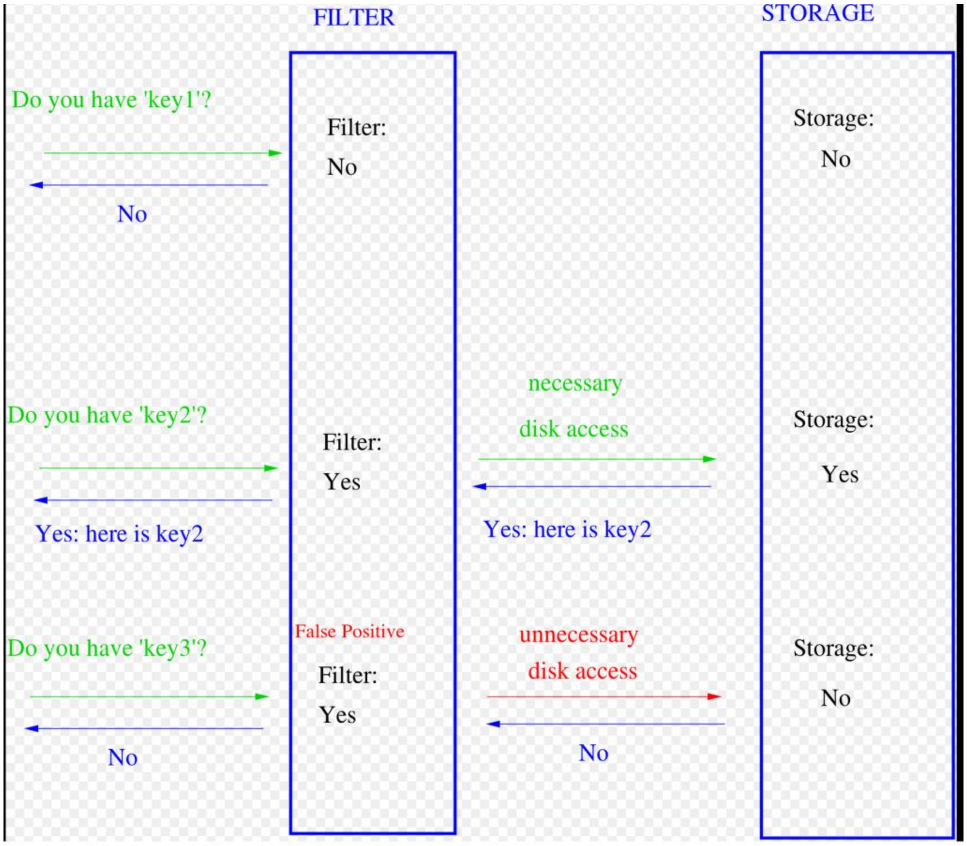
কিন্তু তুমি যদি kk টি পজিশনেই বিট অন পাও তাহলে কিন্তু নিশ্চিতভাবে বলতে পারবে না শব্দটি আছে নাকি নেই। যেমন ধরা যাক “MARY” নামটি হ্যাশ করার পর আমরা পেয়েছি 3, 1 এবং 6। এর আগে ALICE থেকে থেকে পেয়েছিলাম {3, 7, 6} আর BOB থেকে {1, 4, 7}। যখন আমরা MARY খুজবো তখন সবগুলো বিটই অন পাবো, কিন্তু শব্দটা আমরা আগে সেভ করিনি।



এজন্যই আগেই বলেছি ব্লুম ফিল্টার একটা প্রোবাবিলিস্টিক ডাটা স্ট্রাকচার। এটা মাঝে মাঝে False Positive দেয়, অর্থাৎ শব্দটি ডেটাবেজে না থাকলেও বলে যে আছে।

এখন আমরা সার্ভারে প্রতিটা ইউজারনেম হ্যাশ করে ব্লুম ফিল্টার তৈরি করে রাখবো। প্রতিবার ইউজার রেজিস্টার ফর্ম ওপেন করলে আমরা বিট ভেক্টরটি ইউজারের ব্রাউজারে পাঠিয়ে দিবো। ইউজার যখন কোনো নাম টাইপ করবে তখন আমরা সাথে সাথে সেই ফিল্টার চেক করে বলে দিতে পারবো যে ওই নামের কোনো ইউজার ডাটাবেসে আছে নাকি। বেশিরভাগ সময়ই ব্লুম ফিল্টার নেগেটিভ আউটপুট দিবে কারণ নতুন ইউজারনেম সাধারণত অন্য নামের সাথে মিলবে না কিন্তু যদি মিলে যায় তাহলে আমাদের সার্ভারে রিকুয়েস্ট পাঠিয়ে ডাবল-চেক করতে হবে।

নিচের ছবিটি উইকিপিডিয়া থেকে নেয়া:

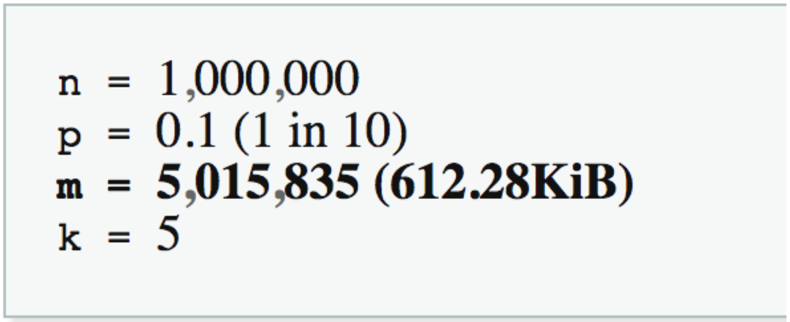


এভাবে ব্লুম ফিল্টার ব্যবহার করে তুমি সার্ভার বা ডাটাবেস এক্সেস অনেক কমিয়ে আনতে পারবে। ব্লুম ফিল্টারের পারফরমেন্স বাড়াতে হলে আমাদেরকে ফলস পজিটিভ কমাতে হবে। ব্লুম ফিল্টারের এরোর রেট মূলত নির্ভর করে হ্যাশ ফাংশনের সংখ্যা (k) এবং ফিল্টারের সাইজের (m) উপর। নিচের ফর্মুলা ব্যবহার করে এরোর রেট বের করা যায়:

$$p \approx (1 - e^{\frac{-kn}{m}})^k$$

এখানে k = হ্যাশ ফাংশনের সংখ্যা, m = বিট ভেক্টরের সাইজ, n = ইনপুটের সাইজ। ফর্মুলা কিভাবে ডিরাইভ করা হয়েছে সেটা আমি এখানে বর্ণনা করবো না, তুমি চাইলে অন্য কোথাও থেকে পড়ে নিতে পারো।

হ্যাশ ফাংশনের সংখ্যা বাড়ালে অথবা m এর সাইজ বাড়ালে এরোর রেট কমে আসবে। কিন্তু তুমি যদি বেশিবার হ্যাশ করবে তত বেশি সময় লাগবে ক্যালকুলেশন করতে, আবার m এর মান বাড়ালে মেমরি বেশি লাগবে। আজকাল অনেক টুলস আছে যেগুলো ব্যবহার করে তুমি এরোর রেট ক্যালকুলেট করতে পারো। যেমন এই টুলটি ব্যবহার করে আমি নিচের ডাটা পেয়েছি:



n = 1,000,000
p = 0.1 (1 in 10)
m = 5,015,835 (612.28KiB)
k = 5

এর মানে আমার যদি ১০ লাখ স্ট্রিং থাকে এবং আমি যদি চাই যে এরোর রেট হবে 0.1% এবং আমি ৫টি হ্যাশ ফাংশন ব্যবহার করবো, তাহলে আমার 612 কিলোবাইট সাইজের ব্লুম ফিল্টার ব্যবহার করতে হবে। দেখতেই পাচ্ছো ব্লুম ফিল্টার খুবই অল্প মেমরি ব্যবহার করে।

ব্লুম ফিল্টারের আরো কিছু ব্যবহার হলো:

- তুমি তোমার সার্ভারে দূর্বল পাসওয়ার্ডের একটা তালিকা রেখে দিতে পারো (যেমন: 123456, password, abcdef)। ইউজার যখন রেজিস্টার করবে এবং নতুন পাসওয়ার্ড দিবে তখন ব্লুম ফিল্টার দিয়ে চেক করতে পারো যে পাসওয়ার্ডটি দূর্বল পাসওয়ার্ডের তালিকায় আছে নাকি।
- ব্লুম ফিল্টার দিয়ে স্পেল-চেকার তৈরি করা যেতে পারে। ইউজারের টাইপ করা শব্দ ডিকশনারিতে না থাকলে ব্লুম ফিল্টার সেটা বলে দিবে।

ব্লুম ফিল্টারের মতো আরেকটা ডাটা-স্ট্রাকচার আছে যেটার নাম “কাউন্ট-মিন স্কেচ” যেটা অনেকটা একই ভাবে কাজ করে কিন্তু প্রয়োগ ভিন্ন, সেটাও এখন শিখে ফেলতে পারো।
আজ এই পর্যন্তই, হ্যাপি কোডিং।