Hashing, 04.12

# Термини

## Същината на проблема

* Като имаме магазин и дойде човек с стока в магазин, да разберем колко и е цената.
* Може линейно да търсим в колекция с продукти и да намерим цената на един продукт.
* Може да ги наредим и да ползваме двоично търсене.
* Може да ги запазим в масив и по някакъв начин да преобразуваме продукт в индекс на масива можем да имаме константен достъп до цената на продукта.
* Масив + hash function = hash table

## Collisions

## Solving collisions

## Load factor

## Важно е с каква сложност е хеширащата функция за performance-a на хеш таблицата

## Check Sum

# Теоретични въпроси

# Hashing function

## Properties

* Deterministic – всеки път при един и същи вход да има един и същ резултат
* Usually not reversible – ако дам резултат от хеш функцията, да не можем да разберем какъв обект е бил подаден на дадената хеш функция
* Know how big your array is – за да знае в какъв range да връща число (да не върне невалиден индекс в масива)
* Should be uniform -
* May only process part of your input – хеша може да се базира само на част от дадения обект, подаден на хеш функцията
* Easy to compute – генерирания хеш да можем лесно да го изпращаме, запазваме…
* Compact – трябва да връща компактен обект.
* В зависимост от сложността на hashing функцията търсенето добавянето и триенето може да не е константно.

## Collisions

* Добрите hashing функции правят равно разпределят обектите на равно разстояние.

### Linear probing

* Клъстерира данните
* Добавяне и търсенето става линейно.
* Има различни видове probing

### Chaining

* Всяка клетка на масива, в който запаметяваме данните е свързан списък и ако имаме колизия добавяме в свързания списък.

### Load factor

* За добра хеш таблица не е важно само добра хеш функция. Колкото повече се пълни масива, толкова по-голяма вероятност за колизии.
* добавени обекти в масива / големина на масива
* добър load factor – 0.5/0.66, ако надмине 0.5/0.66 увеличаваме големината на масива и ре-хешираме всички обекти

### Cuckoo hashing

## Bloom filters

* Ако имаме 10^15 обекта да ги запазим в 10 МБ памет. Може да разгледаме битовете на образуваното число от хеш функцията.
* Правим логическо или на всички хеш битове и със сигурност може да кажем дали няма битовете на някакъв нов обект го няма със 100% сигурност.
* Много данни и искаме да видим дали някакъв нов обект и искаме да кажем дали го няма.
* Поддържаме логическо или на битовете на числата, генерирани от хеш функцията.
* Големината на битовете scale-ва така: броя на елементите \* log(броя на елементите)

## Triple hash

* Dropbox проблема.

## HyperLogLog

* Имаме огромно множество от обекти и искаме да разберем колко е броя на уникалните обекти.
* Хвърляме монета, 4 пъти последователно тура сме получили тура, приблизително е невъзможно да сме хвърлили монетата 4 пъти.
* Търсим последователни единици и им пресмятаме 2 на броя последователни единици.

## Rolling Hash

* Използва концепцията на sliding window
* Rabin karp algorithm
* Имаме един огромен документ, даваме друг голям документ и можем лесно да видим дали един документ се съдържа в друг.

## Perfect Hash

* Хеш функция, която няма колизии, само при крайни множества.

## Хеширане на сложни обекти

* Като имаме сложен обект, му взимаме байтовете и ги хешираме тях.

## Universal family of hash functions

* Пишем си 10 hash функции и после на random избираме. Сменяме хеш функцията, всеки път като сменяме масива.

## Two-way hashing

* Имаме 2 хеш функции, пресмятаме за елемент пресмятаме 2та хеша и виждаме къде има по-малко колизии и слага там.

## Sharding

## The algorithm of dropbox

* Като си качим файл, dropbox го хешира, праща го по мрежата и проверява дали този хеш вече съществува, ако съществува то вашия файл не се качва наново.
* Хеширането се извършва чрез няколко хеш функции.
* Имат няколко сървара и мястото на хеша/файла се базира на деление с остатък от броя на сървърите.

## Distributed hashing

* Всеки един сървър, в един пръстен пази освен своята информация, пази и информацията на двата си съседа.