CS-GY 9223/CS-UY 3943B: Lecture 1
Course Introduction, Randomized Methods, Hashing

NYU Tandon School of Engineering, Prof. Christopher Musco

Algorithmic Machine Learning and Data Science

What is this course all about?

ALGORITHMS IN THE AGE OF DATA SCIENCE

Statistics, machine learning, and data science study how to used data to make better decisions or discoveries. Applications across industry, engineering, and the sciences.

In this class, we study how to do so as quickly as possible, or with bounded computational resources.

APPLICATIONS BY THE NUMBERS

- **Twitter** receives 6,000 tweets every second.
- **Google** receives $\approx$ 10,000 Maps queries every second.
- **NASA** collects 6.4 TB of satallite images every day.
- **Large Synoptic Survey Telescope** will collect 20 TB of images every night.
- **MIT/Harvard Broad Institute** sequences 24 TB of genetic data every day.

ROLE OF ALGORITHMS

Growing demands of data science and machine learning have ushered in a new "golden age" for algorithms research.

- Bolstered by our limited ability to build faster computers (or to access computational resources with a limited financial budget).
- Typical data applications require combining a diverse set of algorithmic tools. Most are not heavily covered in your traditional algorithms curriculum.

CLASS TOPICS

(1) Randomized methods.
(2) Continuous optimization.
(3) Spectral methods and linear algebra.
(4) Fourier methods.

Section 1: Randomized Algorithms.

- Probability tools and concentration of random variables (Markovs, Chebyshev, Chernoff/Bernstein inequalities).
- Random hashing for fast data search, load balancing, and more. Locality sensitive hashing, MinHash, SimHash, etc.
- Sketching and streaming algorithms for compressing and processing data on the fly.
- High-dimensional geometry and the Johnson-Lindenstrauss lemma for compressing high dimensional vectors.

It is hard to find an algorithms paper in 2020 that does not use randomness in some way, but this wasn't always the case!

Section 2: Continuous optimization.

- Gradient descent, stochastic gradient descent, coordinate descent, and how to analyze these methods.
- Acceleration, conditioning, preconditioning, adaptive gradient methods.
- Constrained optimization, linear programming. Ellipsoid and interior point methods.
- Relaxation of combinatorial optimization problems.

Continuous optimization has become the algorithmic workhorse of modern machine learning.

Section 3: Spectral methods and linear algebra.

- How to compute singular value decompositions and eigendecomposition.
- Spectral graph theory: i.e. how to use linear algebara to understanding large graphs through linear algebra (social networks, interaction graphs, etc.).
- Spectral clustering and non-linear dimensionality reduction.
- Random sampling and sketching methods for for matrix computations.

"Complex math operations (machine learning, clustering, trend detection) [are] mostly specified as linear algebra on array data" – Michael Stonebraker, Turing Award Winner

Section 4: Fourier methods.

- Compressed sensing, sparse recovery, and their applications.
- Heavy-hitters/frequent item algorithms for data streams.
- Fourier perspective on machine learning techniques like kernel methods, and the algorithmic benefits.

WHAT WE WON'T COVER

Won't cover:

Software tools or frameworks. MapReduce, Tensorflow, Spark, etc. If you are interested CS-GY 6513 might be a good course.

Machine Learning Models + Techniques. Neural nets, reinforcement learning, Bayesian methods, unsupervised learning etc. I am assuming you already have a course in ML and the focus of this class will be on computational considerations.

But if your research is in machine learning, I think you will find the theoretical tools we learn are more broadly applicable than in designing faster algorithms.

OUR APPROACH

This is primarily a **theory** course.

- Emphasis on proofs of correctness, bounding asymptotic runtimes, convergence analysis, etc. *Why?*
- Learn how to model complex problems in simple ways.
- Learn general mathematical tools that can be applied in a wide variety of problems (in your research, in industry, etc.)
- The homework requires **creative problem solving** and thinking beyond what was covered in class. You will not be able to solve many problems on your first try!

You will need a good background in **probability** and **linear algebra**. See the syllabus for more details. Ask me is you are still unsure of your preparation.

How will you learn all this stuff?

All of this information is on the course webpage `https://www.chrismusco.com/amlds2020/` and in the syllabus posted there! Please read it.

Class structure:

- First half of class (11am - 12pm) will be a flipped course.
- Second half (12:15pm - 1:30pm) will be traditional lecture.
- Short video lecture will be posted on Thursday (must watch on your own time).
- Lecture notes and optional readings posted on course website.

Why?

Class work:

- **Weekly online quiz** (10% of course grade) posted after Thursday lecture. Due following Wednesday before class.
- **Biweekly problem sets** (40% of course grade).
  - These are challenging, and the most effective way to learn the material. I recommend you start early, work with others, ask questions on Piazza, etc.
  - You must write-up solutions on your own.[1]
- **Take-home midterm** (15% of course grade).

[1] 10% bonus on first problem set for using Markdown or LaTex. It should save you time in the long run!

Course project (25% of grade):

- Can be based on a recent algorithms paper. Can be either an experimental or theoretical project.
- Work in pairs.
- We will hold an optional reading group outside of class for those interested in research and getting better at reading and presenting papers. Time TBA.

Class participation project (10% of grade):

- My goal is to know you all individually by the end of this course.
- Lots of ways to earn the full grade: participation in lecture, office hours, or Piazza. Participation in the reading group. Effort on the project.

Important note:

- This is a mixed undergraduate/graduate course.
- Workload is the same, but undergraduates are graded on a different “curve”.
- Tandon Ph.D. student Raphael Meyer will be offering undergraduate only office hours for extra support.

Project Mentor
Aline Besa

Undergrad Mentor
Raphael Meyer

Other important note: No class next week because of memorial day. But I will release a problem set tomorrow and schedule office hours so you can start getting a better sense of the course.

In the next week, please:

- Make sure you are enrolled in the Piazza site.
- Respond to polls for selecting office hour and reading group times.

Questions?

THIS CLASS

Goal: Demonstrate how even the simplest tools from probability can lead to a powerful algorithmic results.

Lecture applications:

- Estimating unique elements from queries.
- Hash tables with worst-case guarantees.
- Load balancing.

Problem set applications:

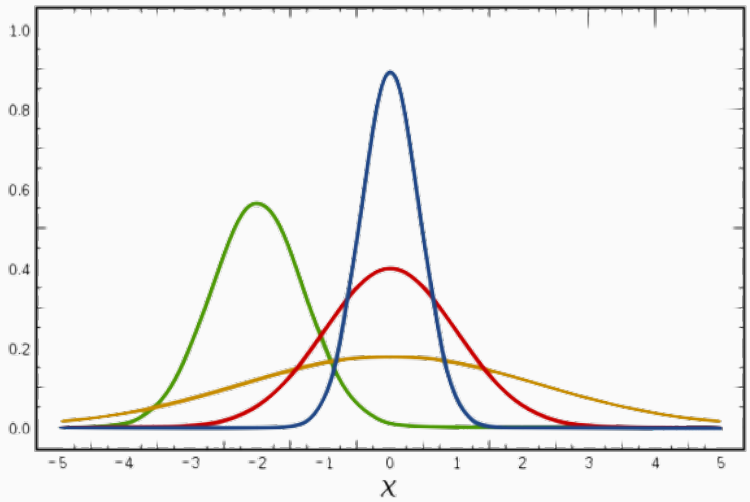
- Group testing for COVID-19.
- Smarter load balancing.

Let X be a random variable taking value in some set $\mathcal{S}$. I.e. for a dice, $\mathcal{S} = \{1, \ldots, 6\}$. For a continuous r.v., we might have $\mathcal{S} = \mathbb{R}$.

- **Expectation**: $\mathbb{E}[X] = \sum_{s \in \mathcal{S}} \Pr[X = s] \cdot s$

  For continuous r.v., $\mathbb{E}[X] = \int_{s \in \mathcal{S}} \Pr(s) \cdot s \, ds$.

- **Variance**: $\mathrm{Var}[X] = \mathbb{E}[(X - \mathbb{E}[X])^2]$



Exercise: For any scalar α, $\mathbb{E}[\alpha X] = \alpha \mathbb{E}[X]$. $\mathrm{Var}[\alpha X] = \alpha^2 \mathrm{Var}[X]$.

Let A and B be random events.

- **Joint Probability**: $\Pr(A \cap B)$. Probability that both events happen.
- **Conditional Probability**: $\Pr(A \mid B) = \frac{\Pr(A \cap B)}{\Pr(B)}$. Probability A happens conditioned on the event that B happens.
- **Independence**: A and B are independent events if: $\Pr(A \mid B) = \Pr(A)$.

Alternative definition of independence:

$$\Pr(A \cap B) = \Pr(A) \cdot \Pr(B).$$

Example: What is the probability that for two independent dice rolls taking value uniformly in $\{1, 2, 3, 4, 5, 6\}$, the first roll comes up odd and the second is < 3?

Let X and Y be random variables. X and Y are independent if, for all events s, t, the random events $[X = s]$ and $[Y = t]$ are independent.

THE MOST POWERFUL THEOREM IN ALL OF PROBABILITY?

Linearity of expectation:

$$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$$

Always, sometimes, or never?

For random variables X, Y:

- $\mathbb{E}[XY] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$.
- $\text{Var}[X + Y] = \text{Var}[X] + \text{Var}[Y]$.
- $\text{Var}[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$.

FIRST APPLICATION

You run a web company that is considering contracting with a vendor that provides CAPTCHAs for logins.

They claim to have a data base of $n = 1,000,000$ unique CAPTCHAs in their database, and a random one will be shown on each API call to their service. They give you access to a test API so you can try it out.

Question: Roughly how many queries to the API, m, would you need to independently verify the claim that there are ~ 1 million unique puzzles?

FIRST APPLICATION

First attempt: Count how many unique CAPTCHAs you see, until you find $1,000,000$ or close to it. Declare that you are satisfied.

As a function of n, roughly how many API queries m do you need?

A DIFFERENT APPROACH

Clever alternative: Count how many duplicate CAPTCHAs you see. If you see the same CAPTCHA on query i and j, that's one duplicate. If you see the same CAPTCHA on queries i, j, and k, that's three duplicates: (i, j), (i, k), (j, k).

If you see many duplicates, the size of the data base probably isn't as big as claimed.

Question: How many duplicates do we expect to see?

Let $D_{i,j} = 1$ if queries i, j return the same CAPTCHA, and 0 otherwise.

This is called an indicator random variable.
$D_{i,j} = \mathbb{1}[\text{CAPTCHA } i \text{ equals CAPTCHA } j]$.

Number of duplicates D is :

$$D = \sum_{i,j \in \{1,\ldots,m\}} D_{i,j}.$$

What is $\mathbb{E}[D]$?

Question: How many duplicates do we expect to see?

Formally, what is $\mathbb{E}[D]$?

$$\mathbb{E}[D] =$$

$n =$ number of CAPTCHAS in database, $m =$ number of test queries.
$D_{i,j} =$ indicator for event CAPTCHA i and j collide.

SOME HARD NUMBERS

Suppose you take $m = 1000$ queries and see 10 duplicates. How does this compare to the expectation if the database actually has $n = 1,000,000$ unique CAPTCHAs?

$$\mathbb{E}[D] = \qquad\qquad = .4995.$$

Something seems wrong... this random variable D came up much larger than it's expectation.

Can we say something formally?

$n =$ number of CAPTCHAS in database, $m =$ number of test queries.

CONCENTRATION INEQUALITY

One of the most important tools in analyzing randomized algorithms. Tell us how likely it is that a random variable X deviates a certain amount from its expectation $\mathbb{E}[X]$.

We will learn three fundamental concentration inequalities that require increasingly stronger assumptions, but provide increasingly tighter results:

1. Markov's Inequality.
2. Chebyshev's Inequality.
3. Hoeffding/Bernstein/Chernoff bounds.

Theorem (Markov's Inequality): For any random variable X which only takes non-negative values any positive t,

$$\Pr[X \geq t] \leq \frac{\mathbb{E}[X]}{t}.$$

Equivalently,

$$\Pr[X \geq \alpha \cdot \mathbb{E}[X]] \leq \frac{1}{\alpha}.$$

Proof:

Suppose you take $m = 1000$ queries and see 10 duplicates. How does this compare to the expectation if the database actually has $n = 1,000,000$ unique CAPTCHAs?

$$\mathbb{E}[D] = \frac{m(m-1)}{2n} = .4995.$$

By Markov's:

$$\Pr[D \geq 10] \leq \frac{\mathbb{E}[D]}{10} < .05 \text{ if } n \text{ actually equals 1 million.}$$

We can be pretty sure we're being scammed...

$n =$ number of CAPTCHAS in database, $m =$ number of test queries.

Alternative view: If $\mathbb{E}[D] = \frac{m(m-1)}{2n}$, then a natural estimator for n is:

$$\tilde{n} = \frac{m(m-1)}{2D}.$$

With a little more work it is possible to show the following:

Claim: If $m = \Omega(\sqrt{n}/\epsilon)$ queries, then with probability 9/10, $(1-\epsilon)n \leq \tilde{n} \leq (1+\epsilon)n$. This is a two-sided multiplicative error guarantee.

This is a lot better than our original method that required $O(n)$ queries!

Fun facts:

- Known as the “mark-and-recapture” method in ecology.
- Can also be used by webcrawlers to estimate the size of the internet, a social network, etc.

This is also closely related to the birthday paradox.

Linearity of Expectation + Markov's Inequality

Primitive but powerful toolkit, which can be applied to a wide variety of applications!

ALGORITHMIC APPLICATION

Goal: Want to store n (key,value) pairs in a data structure, where keys are from a finite by massive universe $\mathcal{U}$. Want to support in $O(1)$ time the operations:

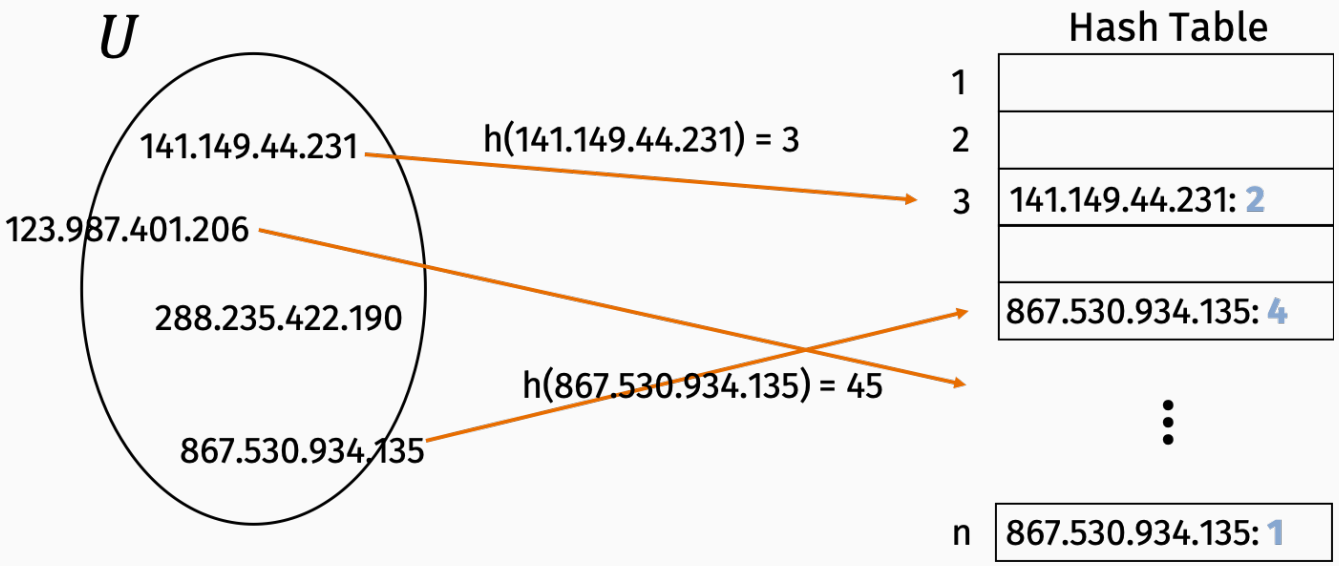
- `insert(key,value)`
- `delete(key)`
- `query(key)`

. **Applications:**

- **URL/DNS resolution**: key is a website url (e.g. www.nyu.edu) and value is an IP address.
- **No-SQL Datastores**: arbitrary keys and values. Amazon DynamoDB, MongoDB, Cassandra.

CLASSIC SOLUTION

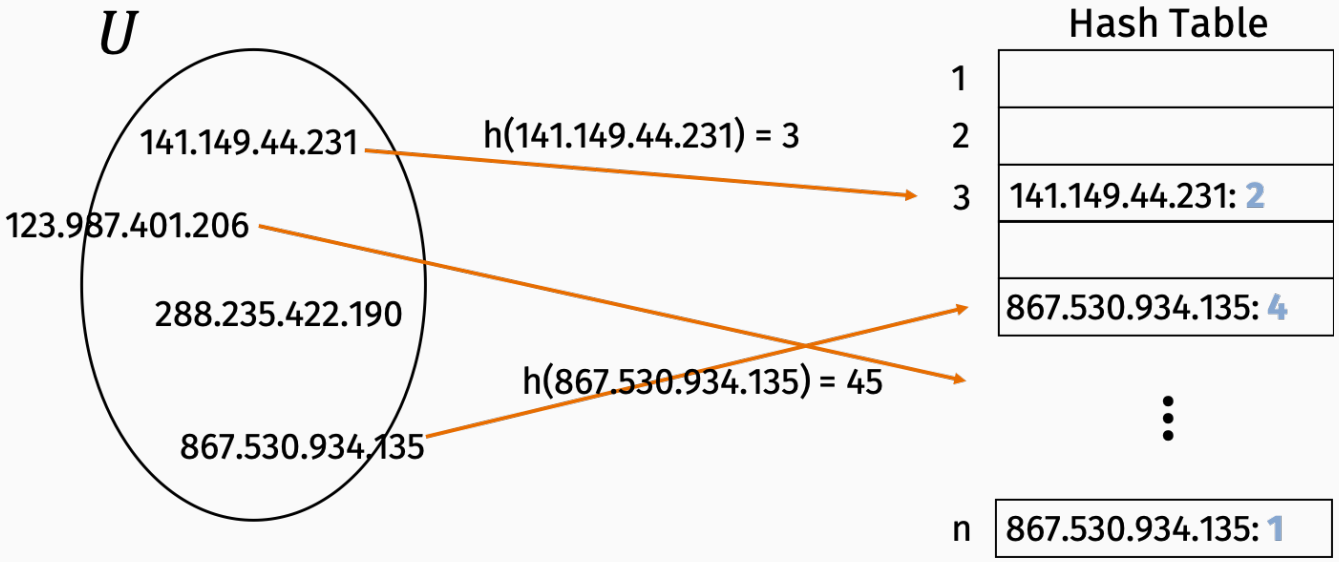
Hash table. Build a table with n slots, and construct a hash function $h : \mathcal{U} \to \{1, \ldots, n\}$.



Example application: a website wants to track how pages each unique visitor (represented by a 128 bit IP address) visits over the course of a day.

Today: Let's only worry about query time, not inserts and deletes..

Typically $|\mathcal{U}| \gg n$, so two keys could hash to the same table slot.



When this happens it's called a **hash collision**. Both pieces of data need to be stored at the same table slot, e.g. in a linked list.

Worse case query time = max number of collisions in a single row.

Question: How can we bound this?

HASH ANALYSIS

If we insert m keys, in the worst case, all could hash to the same bucket, which would yield $O(m)$ worst case query time. We want $O(1)$. To avoid this we either need to assume:

- **Keys are random.** This is hard to guarantee in many applications.
- The hash function is random. This is something we can control!

RANDOM HASH FUNCTION

Let h be a random function from $|\mathcal{U}| \to \{1, \ldots, n\}$. This means that h is chosen using a seed of random numbers, but then the function is fixed. Given input $x \in \mathcal{U}$, it always returns the same output, $h(x)$.

Definition: Uniformly Random Hash Function. A random function $h : \mathcal{U} \to n$ is called uniformly random if:

- $\Pr[h(x) = i] = \frac{1}{n}$ for all $x \in \mathcal{U}$, $i \in \{1, \ldots, n\}$.
- $h(x)$ and $h(y)$ are independent r.v.'s for all $x, y \in \mathcal{U}$.
    - Which implies that $\Pr[h(x) = h(y)] =$

$\mathcal{U} =$ universe of possible keys, $n =$ size of hash table.

RANDOM HASH FUNCTION

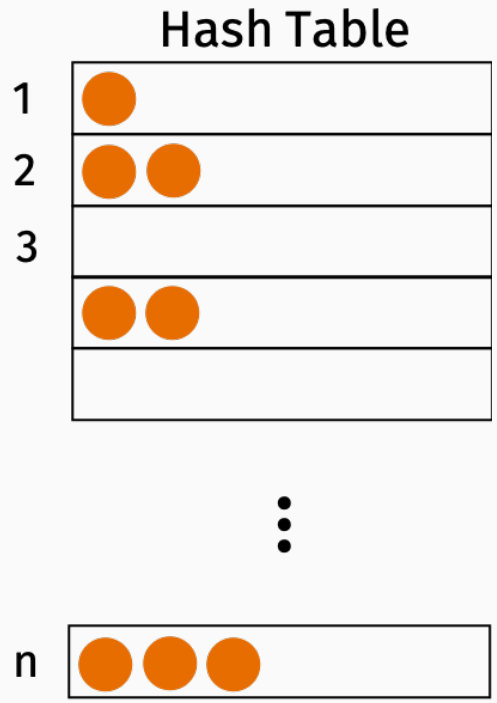
Caveat: It is not possible to efficiently implement uniform random hash functions! But:

- In practice "random looking" functions like MD5, SHA256, etc. often suffice.
- If we have time, we will discuss weaker hash functions (in particular, 2-universal functions) which suffice for our application, and are efficient to implement.

For now, **assume** we have access to a uniformly random hash function. This is an assumption we will use in future lectures as well.

ESTIMATING NUMBER OF HASH COLLISIONS

Let C be the total number of collisions in the hashtable.



What is expected value of C?

m = number of items inserted, n = size of hash table.

ESTIMATING NUMBER OF HASH COLLISIONS

Write down formal expression for C:

Evaluate expectation:

m = number of items inserted, n = size of hash table, h = uniformly random hash function.

COLLISION FREE HASH TABLE

How large does n need to be so that with probability 9/10, there are no collisions at all?

$$\mathbb{E}[C] = \frac{m(m-1)}{2n} = \frac{1}{10}$$

if $n > m^2/5$.

By Markov's inequality,

$$\Pr[C \geq 1] \leq \frac{1}{10},$$

so $C = 0$ with probability 9/10.

Collision free – i.e. $O(1)$ lookup time – with $O(m^2)$ space!

m = number of items inserted, n = size of hash table.

NOTE ON SUCCESS PROBABILITY

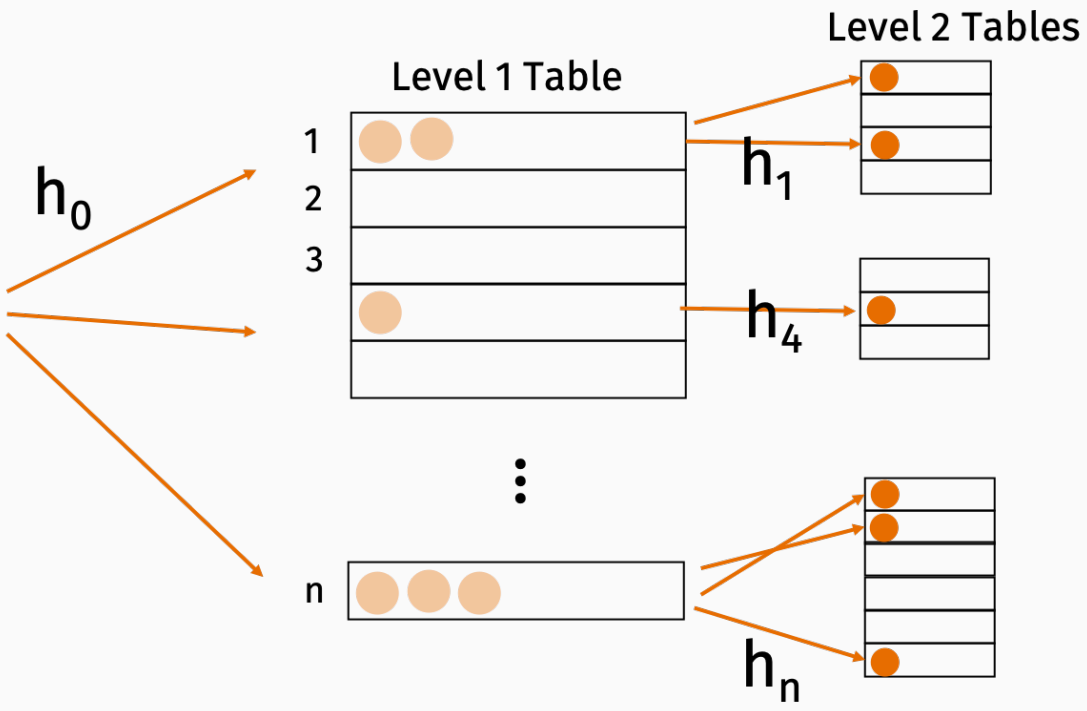
The previous bound holds with high constant probability (9/10). What if we wanted higher probability? Just try different random hash function $h_1, \ldots, h_q$. Probability we find a collision free hash after q tries is:

$$9/10 \quad 99/100 \quad 999/100 \quad \ldots$$

TWO-LEVEL HASHING

$O(m^2)$ is a lot of space to store m items. Can we achieve $O(m)$ space, while maintaining $O(1)$ worst-case lookups?

Key idea: Two-levels of hash tables.



Choose level two tables $T_1, \ldots, T_n$ to be collision-free, which ensures $O(1)$ time lookups.

Let's choose $n = m$ for the Level 1 hash table. How much space is needed for collision-free Level 2 tables?

Let $s_1, \ldots, s_n$ be the number of elements in rows $1, \ldots, n$ of the Level 1 Table.

Total space of Level 2 Tables is:

$$O(s_1^2) + O(s_2^2) + \ldots + O(s_n^2) = O(S),$$

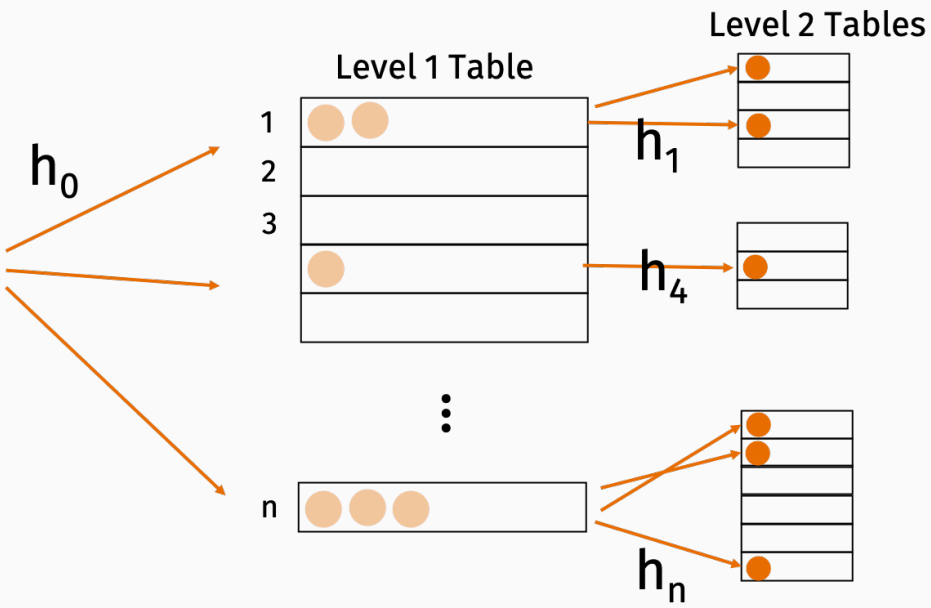
where $S = \sum_{i=1}^{n} s_i^2$.

m = number of items inserted, n = size of level 1 table, C = number of collisions in level 1 table.

TWO-LEVEL HASHING

What is the expected space $\mathbb{E}[S]$?

Claim: $S = 2C + m$.



m = number of items inserted, n = size of level 1 table ($n = m$), C = number of collisions in level 1 table. s_i = number of keys in row i of level 1 table. $S = \sum_{i=1}^{n} s_i^2$.

Result: $\mathbb{E}[S] = 2\mathbb{E}[C] + m =$

Final result: $O(1)$ worst-case hashtable looks-ups using just $O(m)$ space!

m = number of items inserted, n = size of level 1 table ($n = m$), C = number of collisions in level 1 table. s_i = number of keys in row i of level 1 table. $S = \sum_{i=1}^{n} s_i^2$.

Can we weaken our assumption that h is uniformly random?

Definition (2-universal hash function)

A random hash function $h : \mathcal{U} \to \{1, \ldots, n\}$ is 2-universal if, for any fixed $x, y \in \mathcal{U}$,

$$\Pr[h(x) = h(y)] \leq \frac{1}{n}.$$

Claim: A uniformly random hash-function is two universal.

Efficient alternative: Let p be a prime number between $|\mathcal{U}|$ and $2|\mathcal{U}|$. Let a, b be random numbers in $0, \ldots, p$, $a \neq 0$.

$$h(x) = [a \cdot x + b \pmod{p}] \pmod{n}$$

is 2-universal. Lecture notes with proof posted on website.

Another definition you might come across:

Definition (Pairwise independent hash function)

A random hash function $h : \mathcal{U} \to \{1, \ldots, n\}$ is pairwise independent if, for any fixed $x, y \in \mathcal{U}, i, j \in \{1 \ldots, n\}$,

$$\Pr[h(x) = i \cap h(y) = j] = \frac{1}{n^2}.$$

Can we naturally extended to k-wise independence for $k > 2$, which is strictly stronger, and needed for some applications.

Another application of hashing:

Suppose Google answers map search queries using an army of servers $A_1, \ldots, A_q$. Given a query like "new york to rhode island", common practice is to choose a random hash function $h \to \{1 \ldots, q\}$ and to route this query to server:

$$A_{h(\text{“new york to rhode island'})}$$

Why use a hash function instead of just sending the query to a uniformly random server?

Online lecture: We will see how the Markov bound fails us in analyzing this application of hashing, which will motivate a new tool: the Chebyshev inequality.