CSCI 435: Algorithms and Complexity
5. Randomized Algorithms

- *contention resolution*
- *global min cut*
- *linearity of expectation*
- *max 3-satisfiability*
- *universal hashing*

Randomization

Algorithmic design patterns.

- Greedy
- Divide-and-conquer
- Dynamic programming
- Network flow
- *Intractability*
- Randomization

in practice, access to a pseudorandom number generator

Randomization. Allow algorithms to make fair coin flips in unit time.

Why randomize? Can lead to the simplest, fastest, or only known algorithm for a particular problem.

Ex. Symmetry-breaking protocols, graph algorithms, quicksort, hashing, load balancing, closest pair, Monte Carlo integration, cryptography, ...

CSCI 435: Algorithms and Complexity
5. Randomized Algorithms

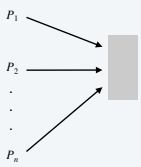
- *contention resolution*
- *global min cut*
- *linearity of expectation*
- *max 3-satisfiability*
- *universal hashing*

Contention resolution in a distributed system

Contention resolution. Given n processes $P_1, \ldots, P_n$, each competing for access to a shared database, devise a protocol to ensure all processes get access on a regular basis.
If two or more processes access the database simultaneously, all processes are locked out.

Restriction. Processes can't communicate.

Challenge. Need a symmetry-breaking paradigm.



Contention resolution: randomized protocol

Protocol. Each process requests access to the database at time t with probability $p = 1/n$.

Claim. Let $S[i, t]$ = event that process i successfully accesses the database at time t. Then $1 / (e \cdot n) \le \Pr[S(i, t)] \le 1/(2n)$.

Pf. By independence, $\Pr[S(i, t)] = p (1 - p)^{n-1}$.

process i requests access; none of remaining n-1 processes request access

- Setting $p = 1/n$, we have $\Pr[S(i, t)] = 1/n \, (1 - 1/n)^{n-1}$. ▪

between 1/e and 1/2; value that maximizes Pr[S(i, t)]

Useful facts from calculus. As n increases from 2, the function:
- $(1 - 1/n)^n$ converges monotonically from $1/4$ up to $1 / e$.
- $(1 - 1/n)^{n-1}$ converges monotonically from $1/2$ down to $1 / e$.

Contention resolution: randomized protocol

Claim. The probability that process i fails to access the database in en rounds is at most $1 / e$. After $e \cdot n \, (c \ln n)$ rounds, the probability is $\le n^{-c}$.

Pf. Let $F[i, t]$ = event that process i fails to access database in rounds 1 through t.
By independence and the previous claim, we have $\Pr[F[i, t]] \le (1 - 1/(en))^t$.

- Choose $t = \lceil e \cdot n \rceil$: $\Pr[F(i, t)] \le \left(1 - \frac{1}{en}\right)^{\lceil en \rceil} \le \left(1 - \frac{1}{en}\right)^{en} \le \frac{1}{e}$
- Choose $t = \lceil e \cdot n \rceil \lceil c \ln n \rceil$: $\Pr[F(i, t)] \le \left(\frac{1}{e}\right)^{c \ln n} = n^{-c}$

Contention resolution: randomized protocol

Claim. The probability that all processes succeed within $2e \cdot n \ln n$ rounds is $\geq 1 - 1/n$.

Pf. Let $F[t]$ = event that at least one of the n processes fails to access the database in any of the rounds 1 through t.

$$\Pr[F[t]] = \Pr\left[\bigcup_{i=1}^{n} F[i,t]\right] \leq \sum_{i=1}^{n} \Pr[F[i,t]] \leq n\left(1-\tfrac{1}{en}\right)^{t}$$

union bound; previous slide

- Choosing $t = 2\lceil en \rceil \lceil c \ln n \rceil$ yields $\Pr[F[t]] \leq n \cdot n^{-2} = 1/n$. ▪

Union bound. Given events $E_1, \ldots, E_n$, $\Pr\left[\bigcup_{i=1}^{n} E_i\right] \leq \sum_{i=1}^{n} \Pr[E_i]$.

CSCI 435: Algorithms and Complexity
5. Randomized Algorithms

- *contention resolution*
- *global min cut*
- *linearity of expectation*
- *max 3-satisfiability*
- *universal hashing*

Global minimum cut

Global min cut. Given a connected, undirected graph $G = (V, E)$, find a cut (A, B) of minimum cardinality.

Applications. Partitioning items in a database, identifying clusters of related documents, network reliability, network design, circuit design, TSP solvers.

Network flow solution.

- Replace every edge (u, v) with two antiparallel edges (u, v) and (v, u).
- Pick some vertex s and compute the min s–v cut separating s from each other node $v \in V$.

Intuition. Global min-cut is harder than min s-t cut. False!

Karger's algorithm

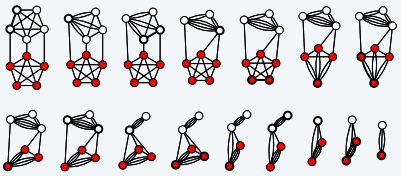
Edge contraction. [Karger, 1995]

- Pick an edge $e = (u, v)$ uniformly at random.
- Contract the edge e.
  - replace u and v by a single new super-node w
  - preserve edges, updating endpoints of u and v to w
  - keep parallel edges, but delete self-loops
- Repeat until the graph has just two nodes u_1 and v_1.
- Return the cut (all nodes that were contracted to form v_1).

Karger's algorithm

Edge contraction. [Karger, 1995]

- Pick an edge $e = (u, v)$ uniformly at random.
- Contract the edge e.
  - replace u and v by a single new super-node w
  - preserve edges, updating endpoints of u and v to w
  - keep parallel edges, but delete self-loops
- Repeat until the graph has just two nodes u_1 and v_1.
- Return the cut (all nodes that were contracted to form v_1).



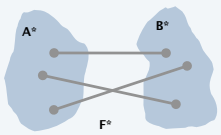
Reference: Thore Husfeldt

Karger's algorithm: analysis

Claim. The contraction algorithm returns a min cut with probability $\geq 2 / n^2$.

Pf. Consider a global min-cut (A^*, B^*) of G.

- Let F^* be the set of edges with one endpoint in A^* and the other in B^*.
- Let $k = |F^*| =$ size of min cut.
- In the first step, the algorithm contracts an edge in F^* with probability $k / |E|$.
- Every node has degree $\geq k$, since otherwise (A^*, B^*) would not be a min cut $\Rightarrow |E| \geq \frac{1}{2} k n \Leftrightarrow k / |E| \leq 2 / n$.
- Thus, the algorithm contracts an edge in F^* with probability $\leq 2 / n$.



Karger's algorithm: analysis

Claim. The contraction algorithm returns a min cut with probability $\geq 2 / n^2$.

Pf. (cont'd)

- Let G' be the graph after j iterations. There are $n' = n - j$ supernodes.
- Suppose no edge in F^* has been contracted. The min cut in G' is still k.
- Since the value of the min cut is k, $|E'| \geq \frac{1}{2} k n' \Leftrightarrow k / |E'| \leq 2 / n'$.
- Thus, the algorithm contracts an edge in F^* with probability $\leq 2 / n'$.
- Let E_j = event that an edge in F^* is not contracted in iteration j.

$$\begin{aligned}
\Pr[E_1 \cap E_2 \cap \cdots \cap E_{n-2}] &= \Pr[E_1] \times \Pr[E_2 \mid E_1] \times \cdots \times \Pr[E_{n-2} \mid E_1 \cap E_2 \cap \cdots \cap E_{n-3}] \\
&\geq \left(1 - \tfrac{2}{n}\right)\left(1 - \tfrac{2}{n-1}\right) \cdots \left(1 - \tfrac{2}{4}\right)\left(1 - \tfrac{2}{3}\right) \\
&= \left(\tfrac{n-2}{n}\right)\left(\tfrac{n-3}{n-1}\right) \cdots \left(\tfrac{2}{4}\right)\left(\tfrac{1}{3}\right) \\
&= \tfrac{2}{n(n-1)} \\
&\geq \tfrac{2}{n^2}
\end{aligned}$$

Karger's algorithm: amplification

Amplification. To amplify the probability of success, run the contraction algorithm many times.

with independent random choices

Claim. If we repeat the contraction algorithm $n^2 \ln n$ times, then the probability of failing to find the global min-cut is $\leq 1 / n^2$.

Pf. By independence, the probability of failure is at most

$$\left(1 - \frac{2}{n^2}\right)^{n^2 \ln n} = \left[\left(1 - \frac{2}{n^2}\right)^{\frac{1}{2}n^2}\right]^{2 \ln n} \leq \left(e^{-1}\right)^{2 \ln n} = \frac{1}{n^2}$$

$(1 - 1/x)^x \leq 1/e$

Karger's algorithm: amplification example

trial 1

trial 2

trial 3

trial 4

trial 5 (finds min cut)

trial 6

...

Reference: Thore Husfeldt

Global min cut: context

Remark. Overall, Karger's algorithm is slow, since we perform $\Theta(n^2 \log n)$ iterations and each iteration takes $\Omega(m)$ time.

Improvement. [Karger–Stein, 1996] $O(n^2 \log^3 n)$.

- Early iterations are less risky than later ones: the probability of contracting an edge in the min cut hits 50% when $n / \sqrt{2}$ nodes remain.
- Repeatedly contract edges until $n / \sqrt{2}$ nodes remain.
- Recursively call the contraction algorithm twice on the resulting graph and return the best of the two cuts.

Extensions. Naturally generalizes to handle positive weights.

Best known. [Karger, 2000] $O(m \log^3 n)$.

faster than best known max flow algorithm or deterministic global min cut algorithm

CSCI 435: Algorithms and Complexity
5. Randomized Algorithms

- *contention resolution*
- *global min cut*
- *linearity of expectation*
- *max 3-satisfiability*
- *universal hashing*

Expectation

Expectation. Given a discrete random variable X, its expectation $E[X]$ is defined by:

$$E[X] = \sum_{j=0}^{\infty} j \Pr[X = j]$$

Waiting for a first success. A coin comes up heads with probability p and tails with probability $1-p$. How many independent flips X should it take until we see our first heads?

$$E[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j] = \sum_{j=0}^{\infty} j (1-p)^{j-1} p = \frac{p}{1-p} \sum_{j=0}^{\infty} j (1-p)^j = \frac{p}{1-p} \cdot \frac{1-p}{p^2} = \frac{1}{p}$$

$j-1$ tails, 1 head

$$\sum_{j=0}^{\infty} j x^j = \frac{x}{(1-x)^2}$$

Expectation: two properties

Useful property. If X is a 0/1 random variable, then $E[X] = \Pr[X = 1]$.

Pf. $E[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j] = \sum_{j=0}^{1} j \cdot \Pr[X = j] = \Pr[X = 1]$

not necessarily independent

Linearity of expectation. Given two random variables X and Y defined over the same probability space, $E[X + Y] = E[X] + E[Y]$.

Benefit. Allows us to decouple a complex calculation into simpler pieces.

Guessing cards

Game. Shuffle a deck of n cards; turn them over one at a time; try to guess each card.

Memoryless guessing. No psychic abilities; can't even remember what's been turned over already. Guess a card from full deck uniformly at random.

Claim. The expected number of correct guesses is 1.

Pf. [surprisingly effortless using linearity of expectation]

- Let $X_i = 1$ if i^{th} prediction is correct, and 0 otherwise.
- Let X = number of correct guesses $= X_1 + \ldots + X_n$.
- $E[X_i] = \Pr[X_i = 1] = 1 / n$.
- $E[X] = E[X_1] + \ldots + E[X_n] = 1/n + \ldots + 1/n = 1$. ▪

linearity of expectation

Guessing cards

Game. Shuffle a deck of n cards; turn them over one at a time; try to guess each card.

Guessing with memory. Guess a card uniformly at random from the cards not yet seen.

Claim. The expected number of correct guesses is $\Theta(\log n)$.

Pf.

- Let $X_i = 1$ if i^{th} prediction is correct, and 0 otherwise.
- Let X = number of correct guesses $= X_1 + \ldots + X_n$.
- $E[X_i] = \Pr[X_i = 1] = 1 / (n - (i - 1))$.
- $E[X] = E[X_1] + \ldots + E[X_n] = 1/n + \ldots + 1/2 + 1/1 = H(n)$. ▪

linearity of expectation

$\ln(n+1) < H(n) < 1 + \ln n$

Collecting prizes

Game. Each box of cereal contains a prize. There are n different types of prizes. Assuming all boxes are equally likely to contain each prize, how many boxes will you open before you have ≥ 1 prize of each type?

Claim. The expected number of boxes is $\Theta(n \log n)$.

Pf.

- Phase j = time between j and $j + 1$ distinct prizes.
- Let X_j = number of boxes you open in phase j.
- Let X = number of boxes in total = $X_0 + X_1 + \ldots + X_{n-1}$.

$$E[X] = \sum_{j=0}^{n-1} E[X_j] = \sum_{j=0}^{n-1} \frac{n}{n-j} = n \sum_{i=1}^{n} \frac{1}{i} = nH(n)$$

probability of success = (n − j) / n
⇒ expected waiting time = n / (n − j)

CSCI 435: Algorithms and Complexity
5. Randomized Algorithms

- *contention resolution*
- *global min cut*
- *linearity of expectation*
- *max 3-satisfiability*
- *universal hashing*

Maximum 3-satisfiability

Maximum 3-satisfiability. Given a 3-SAT formula, find a truth assignment that satisfies as many clauses as possible.

exactly 3 literals per clause and each literal corresponds to a different variable

$$\begin{aligned} C_1 &= x_2 \vee \overline{x_3} \vee \overline{x_4} \\ C_2 &= x_2 \vee x_3 \vee \overline{x_4} \\ C_3 &= \overline{x_1} \vee x_2 \vee x_4 \\ C_4 &= \overline{x_1} \vee \overline{x_2} \vee x_3 \\ C_5 &= x_1 \vee \overline{x_2} \vee \overline{x_4} \end{aligned}$$

Remark. **NP**-hard search problem.

Simple idea. Flip a coin, and set each variable true with probability ½, independently for each variable.

Maximum 3-satisfiability: analysis

Claim. Given a 3-SAT formula with k clauses, the expected number of clauses satisfied by a random assignment is $7k / 8$.

Pf. Consider the random variable $Z_j = \begin{cases} 1 & \text{if clause } C_j \text{ is satisfied} \\ 0 & \text{otherwise.} \end{cases}$

- Let Z = number of clauses satisfied by random assignment. Then

$$\begin{aligned} E[Z] &= \sum_{j=1}^{k} E[Z_j] \\ &= \sum_{j=1}^{k} \Pr[\text{clause } C_j \text{ is satisfied}] \\ &= \tfrac{7}{8} k \end{aligned}$$

linearity of expectation

The probabilistic method

Corollary. For any instance of 3-SAT, there exists a truth assignment that satisfies at least 7/8 of all clauses.

Pf. The random variable is at least its expectation some of the time. ▪

Probabilistic method. [Erdős] Prove the existence of a non-obvious property by showing that a random construction produces it with some positive probability!

Paul Erdős

Maximum 3-satisfiability: analysis

Q. Can we turn this idea into a 7/8-approximation algorithm?
A. Yes (but a random variable can almost always be below its mean).

Lemma. The probability that a random assignment satisfies $\geq 7k / 8$ clauses is at least $1 / (8k)$.

Pf. Let p_j be the probability that exactly j clauses are satisfied;
let p be the probability that $\geq 7k / 8$ clauses are satisfied.

$$\begin{aligned} \tfrac{7}{8} k = E[Z] &= \sum_{j \geq 0} j p_j \\ &= \sum_{j < 7k/8} j p_j + \sum_{j \geq 7k/8} j p_j \\ &\leq (\tfrac{7k}{8} - \tfrac{1}{8}) \sum_{j < 7k/8} p_j + k \sum_{j \geq 7k/8} p_j \\ &\leq (\tfrac{7}{8} k - \tfrac{1}{8}) \cdot 1 + k p \end{aligned}$$

Rearranging terms yields $p \geq 1 / (8k)$. ▪

Maximum 3-satisfiability: analysis

Johnson's algorithm. Repeatedly generate random truth assignments until one of them satisfies $\geq 7k / 8$ clauses.

Theorem. Johnson's algorithm is a 7/8-approximation algorithm.

Pf. By the previous lemma, each iteration succeeds with probability $\geq 1 / (8k)$. By the waiting-time bound, the expected number of trials to find the satisfying assignment is at most $8k$. ▪

Maximum satisfiability

Extensions.

- Allow one, two, or more literals per clause.
- Find a max weighted set of satisfied clauses.

Theorem. [Asano–Williamson, 2000] There exists a 0.784-approximation algorithm for MAX-SAT.

Theorem. [Karloff–Zwick, 1997; Zwick+computer, 2002] There exists a 7/8-approximation algorithm for version of MAX-3-SAT in which each clause has at most 3 literals.

Theorem. [Håstad, 1997] Unless **P** = **NP**, no ρ-approximation algorithm for MAX-3-SAT (and hence MAX-SAT) for any $\rho > 7/8$.

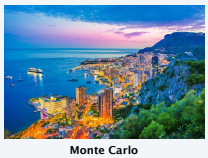
very unlikely to improve over simple randomized algorithm for MAX-3-SAT

Types of randomized algorithms

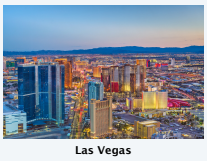
Monte Carlo. Guaranteed to run in poly-time, likely to find correct answer.
Ex. Karger's algorithm for global min cut.

Las Vegas. Guaranteed to find correct answer, likely to run in poly-time.
Ex. Randomized quicksort, Johnson's MAX-3-SAT algorithm.

Remark. One can always convert a Las Vegas algorithm into Monte Carlo, but there is no known method (in general) to convert the other way.



Monte Carlo



Las Vegas

Randomized complexity theory

RP. [Monte Carlo] The class of decision problems that are solvable with one-sided error in polynomial time.

One-sided error. (can decrease probability of false negative to be arbitrarily small via amplification)
- If the correct answer is *no*, always return *no*.
- If the correct answer is *yes*, return *yes* with probability ≥ ½.

ZPP. [Las Vegas] The class of decision problems that are solvable in expected polynomial time.

(running time can be unbounded, but fast on average)

Theorem. **P** ⊆ **ZPP** ⊆ **RP** ⊆ **NP**.

Fundamental open questions. To what extent does randomization help? Does **P** = **ZPP** ? Does **ZPP** = **RP** ? Does **RP** = **NP** ?

CSCI 435: Algorithms and Complexity
5. Randomized Algorithms

- *contention resolution*
- *global min cut*
- *linearity of expectation*
- *max 3-satisfiability*
- *universal hashing*

Dictionary ADT

Dictionary. Given a universe U of possible elements, maintain a subset $S \subseteq U$ so that inserting, deleting, and searching in S is efficient.

Dictionary interface.
- *create*(): initialize a dictionary with $S = \emptyset$.
- *insert*(u): add element $u \in U$ to S.
- *delete*(u): delete u from S (if u is currently in S).
- *lookup*(u): is u in S ?

Challenge. The universe U can be extremely large, so defining an array of size $|U|$ is infeasible.

Applications. File systems, databases, Google, compilers, checksums, P2P networks, associative arrays, cryptography, web caching, etc.

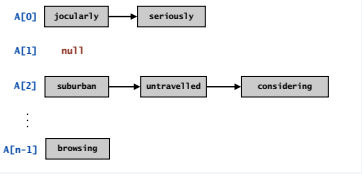
Hashing

Hash function. $h : U \to \{ 0, 1, \ldots, n-1 \}$.

Hashing. Create an array A of length n. When processing an element u, access the array element $A[h(u)]$.

Collision. Occurs when $h(u) = h(v)$, but $u \neq v$. (birthday paradox)

- A collision is expected after $\Theta(\sqrt{n})$ random insertions.
- Separate chaining: $A[i]$ holds a linked list of elements u with $h(u) = i$.



A first attempt at hashing

Ad-hoc hash function.

```
HASH (s, n)
h ← 0
FOR (0 ≤ i ≤ |s|)
   h ← 31 × h + s[i]
RETURN h mod n
```

hash function à la Java string library

Deterministic hashing. If $|U| \geq n^2$, then for any fixed hash function h, there is a subset $S \subseteq U$ of n elements that all hash to the same slot. Thus, $\Theta(n)$ time per lookup in worst-case.

Q. Isn't the ad-hoc hash function good enough in practice?

Algorithmic attacks

When can't we live with the ad-hoc hash function?

- Obvious situations: aircraft control, nuclear reactors, pacemakers, ...
- Surprising situations: denial-of-service (DOS) attacks.

malicious adversary learns your ad-hoc hash function and causes a big pile-up in a single slot that grinds performance to a halt

Real world exploits. [Crosby–Wallach, 2003]

- Perl 5.8.0: insert carefully chosen strings into an associative array.
- Bro intrusion detection system: send carefully chosen packets to slow down the software's internal hash table.

File version	Perl 5.6.1 program	Perl 5.8.0 program
Perl 5.6.1	6506 seconds	<2 seconds
Perl 5.8.0	<2 seconds	6838 seconds

Table 1: CPU time inserting 90k short attack strings into two versions of Perl.

	Attack	Random
Total CPU time	44.50 min	.86 min
Hash table time	43.78 min	.02 min

Table 2: Total CPU time and CPU time spent in hash table code during an offline processing run of 64k attack and 64k random SYN packets.

Denial of Service via Algorithmic Complexity Attacks

Scott A. Crosby
scrosby@cs.rice.edu
Dan S. Wallach
dwallach@cs.rice.edu
Department of Computer Science, Rice University

A better attempt at hashing

Ideal hash function. Map m elements uniformly at random to n hash slots.

- Running time depends on the length of chains.
- Average length of chain = $\alpha = m / n$.
- Choose $n \approx m \Rightarrow$ expect $O(1)$ per insert, lookup, or delete.

Challenge. Find an explicit hash function h that achieves $O(1)$ per operation.

Approach. Use randomization for the choice of h.

adversary knows the randomized algorithm you're using, but doesn't know the random choice that the algorithm makes

Universal hashing

A universal family of hash functions is a set of hash functions H mapping a universe U to the set $\{ 0, 1, \ldots, n-1 \}$ such that

- For any pair of elements $u \neq v$, $\Pr_{h \in H}[h(u) = h(v)] \leq 1/n$.
- Can select random h efficiently.
- Can compute $h(u)$ efficiently.

chosen uniformly at random

Ex. $U = \{ a, b, c, d, e, f \}$, $n = 2$.

	a	b	c	d	e	f
$h_1(x)$	0	1	0	1	0	1
$h_2(x)$	0	0	0	1	1	1

$H = \{h_1, h_2\}$

$\Pr_{h \in H}[h(a) = h(b)] = 1/2$

$\Pr_{h \in H}[h(a) = h(c)] = 1$

$\Pr_{h \in H}[h(a) = h(d)] = 0$

not universal

	a	b	c	d	e	f
$h_1(x)$	0	1	0	1	0	1
$h_2(x)$	0	0	0	1	1	1
$h_3(x)$	0	0	1	0	1	1
$h_4(x)$	1	0	0	1	1	0

$H = \{h_1, h_2, h_3, h_4\}$

$\Pr_{h \in H}[h(a) = h(b)] = 1/2$

$\Pr_{h \in H}[h(a) = h(c)] = 1/2$

$\Pr_{h \in H}[h(a) = h(d)] = 1/2$

$\Pr_{h \in H}[h(a) = h(e)] = 1/2$

$\Pr_{h \in H}[h(a) = h(f)] = 0$

universal

Universal hashing: analysis

Proposition. Let H be a universal family of hash functions mapping a universe U to the set $\{ 0, 1, \ldots, n-1 \}$, let $h \in H$ be chosen uniformly at random from H, let $S \subseteq U$ be a subset of size at most n, and let $u \notin S$. Then the expected number of items in S that collide with u is at most 1.

Pf. For any $s \in S$, define a random variable $X_s = 1$ if $h(s) = h(u)$ and 0 otherwise. Let X be a random variable counting the total number of collisions with u. Then

$$E_{h \in H}[X] = E[\sum_{s \in S} X_s] = \sum_{s \in S} E[X_s] = \sum_{s \in S} \Pr[X_s = 1] \leq \sum_{s \in S} \frac{1}{n} = |S| \frac{1}{n} \leq 1$$

linearity of expectation

X_s is a 0–1 random variable

universal

Q. Okay, now how do we design a universal class of hash functions?

Universal hashing: design

Modulus. We will use a prime number p for the size of the hash table.

Integer encoding. Identify each element $u \in U$ with a base-p integer of r digits: $x = (x_1, x_2, \ldots, x_r)$.

Hash function. Let A = set of all r-digit, base-p integers. For each $a = (a_1, a_2, \ldots, a_r)$ where $0 \le a_i < p$, define

$$h_a(x) = \left(\sum_{i=1}^{r} a_i x_i \right) \bmod p$$ ← maps universe U to set $\{ 0, 1, \ldots, p-1 \}$

Hash function family. $H = \{ h_a : a \in A \}$.

Universal hashing: design

Theorem. $H = \{ h_a : a \in A \}$ is a universal family of hash functions.

Pf. Let $x = (x_1, x_2, \ldots, x_r)$ and $y = (y_1, y_2, \ldots, y_r)$ be two distinct elements of U.

We need to show that $\Pr[h_a(x) = h_a(y)] \le 1 / p$.

- Since $x \ne y$, there exists an integer j such that $x_j \ne y_j$.
- We have $h_a(x) = h_a(y)$ iff $a_j \underbrace{(y_j - x_j)}_{z} \equiv \underbrace{\sum_{i \ne j} a_i (x_i - y_i)}_{m} \bmod p$.
- We can assume a was chosen uniformly at random by first selecting all coordinates a_i where $i \ne j$, then selecting a_j at random. Thus, we can assume a_i is fixed for all coordinates $i \ne j$.
- Since p is prime, $a_j z \equiv m \bmod p$ has at most one solution among p possibilities. ← number theory fact
- Thus $\Pr[h_a(x) = h_a(y)] \le 1 / p$. ▪

Universal hashing

Goal. Given a universe U, maintain a subset $S \subseteq U$ so that insertion, deletion, and lookup are efficient.

Universal hash function family. $H = \{ h_a : a \in A \}$.

$$h_a(x) = \left(\sum_{i=1}^{r} a_i x_i \right) \bmod p$$

- Choose p prime so that $m \le p \le 2m$, where $m = |S|$.
- Fact. There exists a prime between m and $2m$. ← we can find such a prime using another randomized algorithm (!)

Consequences.

- Space used = $\Theta(m)$.
- Expected number of collisions per operation is ≤ 1
  $\Rightarrow$ $O(1)$ time per insertion, deletion, or lookup.