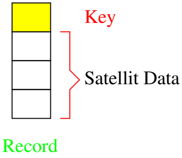
Hashing

AiC FME, UPC

Fall 2023

Data Structures: Reminder

Given a universe $\mathcal{U}$, a dynamic set of records, where each record:



- Array
- Linked List (and variations)
- Stack (LIFO): Supports push and pop
- Queue (FIFO): Supports enqueue and dequeue
- Deque: Supports push, pop, enqueue and dequeue
- Heaps: Supports insertions, deletions, find Max and MIN
- Hashing

Data structures for dynamic sets

DICTIONARY

Data structure for maintaining $\mathcal{S} \subset \mathcal{U}$ together with operations:

- Search(k): decide if $k \in \mathcal{S}$
- Insert(k): $\mathcal{S} := \mathcal{S} \cup \{k\}$
- Delete(k): $\mathcal{S} := \mathcal{S} \backslash \{k\}$

PRIORITY QUEUE

Data structure for maintaining $\mathcal{S} \subset \mathcal{U}$ together with operations:

- Insert(x, k): $\mathcal{S} := \mathcal{S} \cup \{x\}$
- Maximum(): Returns element of $\mathcal{S}$ with largest key value
- Extract-Maximum(): Returns (x, k) with k largest value in $\mathcal{S}$, $\mathcal{S} = \mathcal{S} - \{x\}$.

Priority Queue implementations

Linked List:

- *INSERT:* $O(n)$
- *EXTRACT-MAX:* $O(1)$

Heap:

- *INSERT:* $O(\lg n)$
- *EXTRACT-MAX:* $O(\lg n)$

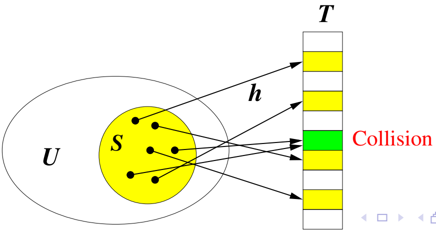
Using a Heap is a good compromise between fast insertion and slow extraction.

Hashing

Data Structure that supports *dictionary* operations on an universe of numerical keys.

- Notice the number of possible keys represented as 64-bit integers is $2^{63} = 18446744073709551616$.
- Tradeoff *time/space*
- Define a hashing table $T[0, \dots, m-1]$
- a hashing function $h : \mathcal{U} \to T[0, \dots, m-1]$

Hans P. Luhn
(1896-1964)

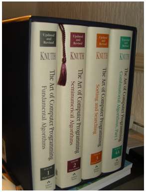


Simple uniform hashing function.

- We want to store a maximum of n keys in a hashing table T with m slots.
- The performance of hashing depends on how well h distributes the keys on the m slots.
- h is simple uniform if it hash any key *with equal probability* into any slot, independently of where other keys go.
- In this way, we get a load factor $\alpha = n/m$, the average number of keys per slot.

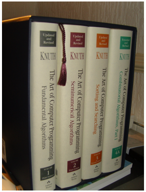
How to choose h?

Advice: For an exhaustive treaty on Hashing: D. Knuth, Vol. 3 of *The Art of computing programming*



How to choose h?

Advice: For an exhaustive treaty on Hashing: D. Knuth, Vol. 3 of *The Art of computing programming*



h depends on the type of key:

- For keys in the real interval $[0, 1)$, we can use $h(k) = \lfloor mk \rfloor$.
- For keys in the real interval $[s, t)$ scale by $1/(t-s)$, and use the previous method, $h(k/(t-s)) = \lfloor mk/(t-s) \rfloor$.

The division method

Choose m prime or as far as possible from a power of 2,

$$\boxed{h(k) = k \mod m}.$$

Fast ($\Theta(1)$) to compute in most languages ($k\%m$)!

Be aware: if $m = 2^r$ the hash does not depend on all the bits of K

If $r = 6$ with $k = 1011000111\underbrace{011010}_{=h(k)}$

$(45530 \mod 64 = 858 \mod 64)$

In some applications, the keys may be very large, for instance with alphanumeric keys, which must be converted to ascii, and reinterpreted as numbers in binary.

Example: *averylongkey* is converted via ascii:

$$97 \cdot 128^{11} + 118 \cdot 128^{10} + 101 \cdot 128^{9} + 114 \cdot 128^{8} + 121 \cdot 128^{7} + 108 \cdot 126^{6} + 111 \cdot 128^{5} + 110 \cdot 128^{4} + 103 \cdot 128^{3} + 107 \cdot 128^{2} + 101 \cdot 128^{1} + 121 \cdot 128^{0} = n$$

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

which has 84-bits!

How to deal with large n?

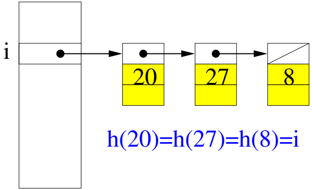
For large n, to compute $h = n \mod m$, we can use mod arithmetic + Horner's method:

$$
\begin{aligned}
&((((((((((97 \cdot 128 + 118) \cdot 128 + 101) \cdot 128 + 114) \cdot 128 + 121) \\
&\cdot 128 + 111) \cdot 128 + 110) \cdot 128 + 103) \cdot 128 + 107) \\
&\cdot 128 + 101) \cdot 128 + 121 \mod m \\
&= (((((((((\underbrace{\underbrace{(97 \cdot 128 + 118 \mod m)}_{} \cdot 128) \mod m + 101)}_{} \cdot \ldots))))))
\end{aligned}
$$

Collision resolution: Separate chaining

For each table address, construct a linked list of the items whose keys hash to that address.

- Every key goes to the same slot
- Time to explore the list = length of the list



Cost of average analysis of chaining

The cost of the dictionary operations using hashing:

- Insertion of a new key: $\Theta(1)$.
- Search of a key: $O($ ength of the list)
- Deletion of a key: $O(\text{length of the list})$.

Under the hypothesis that h is *simply uniform hashing*, each key x is equally likely to be hashed to any slot of T, independently of where other keys are hashed

Therefore, the expected number of keys falling into $T[i]$ is $\alpha = n/m$.

Cost of search

- For an unsuccessful search (x is not in T), we have to explore the ist at $h(x) \rightarrow T[i]$. So, the expected time to search the list at $T[i]$ is $O(1+\alpha)$.
  (α of searching the list and $\Theta(1)$ of computing $h(x)$ and going to slot $T[i]$)

Cost of search

- For an unsuccessful search (x is not in T), we have to explore the ist at $h(x) \to T[i]$. So, the expected time to search the list at $T[i]$ is $O(1 + \alpha)$.
  (α of searching the list and $\Theta(1)$ of computing $h(x)$ and going to slot $T[i]$)
- For an successful search, we obtain the same bound, although in most of the cases we would have to search a fraction of the list until finding the x element.)

Cost of search

- For an unsuccessful search (x is not in T), we have to explore the ist at $h(x) \to T[i]$. So, the expected time to search the list at $T[i]$ is $O(1+\alpha)$.
  (α of searching the list and $\Theta(1)$ of computing $h(x)$ and going to slot $T[i]$)
- For an successful search, we obtain the same bound, although in most of the cases we would have to search a fraction of the list until finding the x element.)
- Under the assumption of simple uniform hashing, in a hash table with chaining, a search takes time $\Theta(1+\frac{n}{m})$ on average.

Cost of search

- For an unsuccessful search (x is not in T), we have to explore the ist at $h(x) \to T[i]$. So, the expected time to search the list at $T[i]$ is $O(1+\alpha)$.
  (α of searching the list and $\Theta(1)$ of computing $h(x)$ and going to slot $T[i]$)
- For an successful search, we obtain the same bound, although in most of the cases we would have to search a fraction of the list until finding the x element.)
- Under the assumption of simple uniform hashing, in a hash table with chaining, a search takes time $\Theta(1+\frac{n}{m})$ on average.
- Notice that if $n = \theta(m)$ then $\alpha = O(1)$ and search time is $\Theta(1)$.

Universal hashing: Motivation

- For every deterministic hash function, there is a set of bad instances.
- An adversary can arrange the keys so your function hashes most of them to the same slot.

Universal hashing: Motivation

- For every deterministic hash function, there is a set of bad instances.
- An adversary can arrange the keys so your function hashes most of them to the same slot.
- Create a set $\mathcal{H}$ of hash functions on $\mathcal{U}$ and choose a hashing function at random and independently of the keys.
- The adversary might known the probability space but not the particular selection.

Universal hashing

Let $\mathcal{U}$ be the universe of keys and let $\mathcal{H}$ be a collection of hashing functions with hashing table $T[0,\ldots,m-1]$, $\mathcal{H}$ is universal if $\forall x,y \in \mathcal{U}, x \neq y$, then

$$|\{h \in \mathcal{H} \mid h(x) = h(y)\}| \leq \frac{|\mathcal{H}|}{m}.$$

In an equivalent way, $\mathcal{H}$ is *universal* if $\forall x,y \in \mathcal{U}, x \neq y$, and for any h chosen uniformly from $\mathcal{H}$, we have

$$\mathbf{Pr}\left[h(x) = h(y)\right] \leq \frac{1}{m}.$$

Universality gives good average-case behaviour

Theorem

If we pick u.a.r. h from a universal family $\mathcal{H}$ and build a table with size m for a set of n keys, for any given key x let C_x be a random variable counting the number of collisions with others keys y in T.

$$\mathbf{E}\left[C_x\right] \leq n/m.$$

Construction of a universal family: $\mathcal{H}$

Let $\mathcal{U}$ be the key universe and let N be the maximum key value. Our target is a hash table with m positions, $T[0,\ldots,m-1]$.

- *Choose a prime p, $N \leq p \leq 2N$. Then $\mathcal{U} \subset \mathbb{Z}_p = \{0,1,\ldots,p-1\}$.*
- Define $\mathcal{H} = \{h_{a,b} | a,b \in \mathbb{Z}_p, a \neq 0\}$.

Construction of a universal family: $\mathcal{H}$

Let $\mathcal{U}$ be the key universe and let N be the maximum key value. Our target is a hash table with m positions, $T[0, \dots, m-1]$.

- *Choose a prime p, $N \leq p \leq 2N$. Then $\mathcal{U} \subset \mathbb{Z}_p = \{0, 1, \dots, p-1\}$.*
- Define $\mathcal{H} = \{h_{a,b} | a, b \in \mathbb{Z}_p, a \neq 0\}$.
- To select u.a.r. $h \in \mathcal{H}$, *choose independently and u.a.r. $a \in \mathbb{Z}_p^+$ and $b \in \mathbb{Z}_p$. Given a key x define* $h_{a,b}(x) = \underbrace{((ax+b) \mod p)}_{g_{a,b}(x)} \mod m$.

Construction of a universal family: $\mathcal{H}$

Let $\mathcal{U}$ be the key universe and let N be the maximum key value. Our target is a hash table with m positions, $T[0, \ldots, m-1]$.

- *Choose a prime p, $N \leq p \leq 2N$. Then $\mathcal{U} \subset \mathbb{Z}_p = \{0, 1, \ldots, p-1\}$.*
- Define $\mathcal{H} = \{h_{a,b} | a, b \in \mathbb{Z}_p, a \neq 0\}$.

- To select u.a.r. $h \in \mathcal{H}$, *choose independently and u.a.r. $a \in \mathbb{Z}_p^+$ and $b \in \mathbb{Z}_p$. Given a key x define $h_{a,b}(x) = \underbrace{((ax+b) \mod p)}_{g_{a,b}(x)} \mod m$.*

- Example: $p = 17, m = 6$, we have $\mathcal{H}_{17,6} = \{h_{a,b} : a \in \mathbb{Z}_p^+, b \in \mathbb{Z}_p\}$
  if $x = 8$, $a = 3, b = 4$ then
  $h_{3,4}(8) = ((3 \cdot 8 + 4) \mod 17) \mod 6 = 5$

Properties of $\mathcal{H}$

1. $h_{ab} : \mathbb{Z}_p \to \mathbb{Z}_m$.
2. $|\mathcal{H}| = p(p-1)$. (We can select a in $p-1$ ways and b in p ways)
3. Specifying an $h \in \mathcal{H}$ requires $O(\lg p) = O(\lg N)$ bits.
4. To choose $h \in \mathcal{H}$ select a, b independently and u.a.r. from $\mathbb{Z}_p^+$ and $\mathbb{Z}_p$.
5. Evaluating $h(x)$ is fast.

Theorem

The family $\mathcal{H}$ is universal.

For the proof:

Chapter 11 of Cormen. Leiserson, Rivest, Stein: *An introduction to Algorithms*

Markov's inequality

Lemma (Markov's inequality)

If $X \geq 0$ is a r.v, for any constant $a > 0$,

$$\mathbf{Pr}\left[X \geq a\right] \leq \frac{\mathbf{E}\left[X\right]}{a}.$$

Markov's inequality

Lemma (Markov's inequality)

If $X \geq 0$ *is a r.v, for any constant* $a > 0$,

$$\mathbf{Pr}\left[X \geq a\right] \leq \frac{\mathbf{E}\left[X\right]}{a}.$$

Corollary

If $X \geq 0$ *is a r.v, for any constant* $b > 0$,

$$\mathbf{Pr}\left[X \geq b\,\mathbf{E}\left[X\right]\right] \leq \frac{1}{b}.$$

Chebyshev's Inequality

Pafnuty Chebyshev (XIXc)

If you can compute the **Var** [] then you can compute σ and get better bounds for concentration of any r.v. (positive or negative).

Theorem

Let X be a r.v. with expectation μ and standard deviation $\sigma > 0$, then for any $a > 0$

$$\mathbf{Pr}\left[|X - \mu| \geq a\,\sigma\right] \leq \frac{1}{a^2}.$$

Note that $|X - \mu| \geq a\sigma \Leftrightarrow (X \geq a\sigma + \mu) \cup (X \geq \mu - a\sigma)$.

Chernoff Bounds

Sergei Bernstein (1924), Wassily Hoeffding (1964),
Herman Chernoff (1952)

The Chernoff bound can be used when the random variable X is the sum of several independent Poisson trials, where each X_i can has probability of success p_i. The particular case where all p_i are equal is the Bernouilli trials.

Theorem ((Ch-1))

Let $\{X_i\}_{i=0}^{n}$ be independent Poisson trials, with $\mathbf{Pr}\left[X_i = 1\right] = p_i$. Then, if $X = \sum_{i=1}^{n} X_i$, and $\mu = \mathbf{E}\left[X\right]$, we have

1. $\mathbf{Pr}\left[X \leq (1-\delta)\mu\right] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\mu}$, *for* $\delta \in (0,1)$.
2. $\mathbf{Pr}\left[X \geq (1+\delta)\mu\right] \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu}$ *for any* $\delta > 0$.

Weak Chernoff's bound, but easy to use

Corollary (Ch-2)

Let $\{X_i\}_{i=0}^n$ *be independent Poisson trials, with* $\mathbf{Pr}[X_i = 1] = p_i$*. Then if* $X = \sum_{i=1}^n X_i$*, and* $\mu = \mathbf{E}[X]$*, we have*

1. $\mathbf{Pr}[X \leq (1-\delta)\mu] \leq e^{-\mu\delta^2/2}$*, for* $\delta \in (0,1)$*.*
2. $\mathbf{Pr}[X \geq (1+\delta)\mu] \leq e^{-\mu\delta^2/3}$*, for* $\delta \in (0,1]$*.*

An immediate corollary to the previous result:

Corollary (Ch-3)

Let $\{X_i\}_{i=0}^n$ *be independent Poisson trials, with* $\mathbf{Pr}[X_i = 1] = p_i$*. Then if* $X = \sum_{i=1}^n X_i$*,* $\mu = \mathbf{E}[X]$ *and* $\delta \in (0,1)$*, we have*

$$\mathbf{Pr}[|X - \mu| \geq \delta\mu] \leq 2e^{-\mu\delta^2/3}.$$

Counting the number of distinct elements

Counting the number of distinct elements

- Distinct elements problem: Given a stream s, output $|\{j \mid f_j > 0\}|$. where f_j is the frequency of the j in the stream s

Counting the number of distinct elements

- Distinct elements problem: Given a stream s, output $|\{j \mid f_j > 0\}|$. where f_j is the frequency of the j in the stream s
- In order to solve the problem using sublinear space, we need to use probabilistic algorithms/data structure and some adequate notion of approximation.

An (ϵ, δ)-approximation

An (ϵ, δ)-approximation

- Let $\mathcal{A}(s)$ denote the output of a randomized streaming algorithm $\mathcal{A}$ on input s; note that this is a random variable.
- Let $\Phi(s)$ be the function that $\mathcal{A}$ is supposed to compute.

An (ϵ, δ)-approximation

- Let $\mathcal{A}(s)$ denote the output of a randomized streaming algorithm $\mathcal{A}$ on input s; note that this is a random variable.
- Let $\Phi(s)$ be the function that $\mathcal{A}$ is supposed to compute.
- $\mathcal{A}$ is a (ϵ, δ)-approximation to Φ if we have

$$Pr\left[\left|\frac{\mathcal{A}(s)}{\Phi(s)} - 1\right| > \epsilon\right] \leq \delta.$$

An (ϵ, δ)-approximation

- Let $\mathcal{A}(s)$ denote the output of a randomized streaming algorithm $\mathcal{A}$ on input s; note that this is a random variable.
- Let $\Phi(s)$ be the function that $\mathcal{A}$ is supposed to compute.
- $\mathcal{A}$ is a (ϵ, δ)-approximation to Φ if we have

$$Pr\left[\left|\frac{\mathcal{A}(s)}{\Phi(s)} - 1\right| > \epsilon\right] \leq \delta.$$

- $\mathcal{A}$ is a (ϵ, δ)-additive approximation to Φ if we have

$$Pr\left[|\mathcal{A}(s) - \Phi(s)| > \epsilon\right] \leq \delta.$$

An (ϵ, δ)-approximation

- Let $\mathcal{A}(s)$ denote the output of a randomized streaming algorithm $\mathcal{A}$ on input s; note that this is a random variable.
- Let $\Phi(s)$ be the function that $\mathcal{A}$ is supposed to compute.
- $\mathcal{A}$ is a (ϵ, δ)-approximation to Φ if we have

$$Pr\left[\left|\frac{\mathcal{A}(s)}{\Phi(s)} - 1\right| > \epsilon\right] \leq \delta.$$

- $\mathcal{A}$ is a (ϵ, δ)-additive approximation to Φ if we have

$$Pr\left[|\mathcal{A}(s) - \Phi(s)| > \epsilon\right] \leq \delta.$$

- When $\delta = 0$, $\mathcal{A}$ must be deterministic.
  When $\epsilon = 0$, $\mathcal{A}$ must be an exact algorithm.

Counting the number of distinct elements

Counting the number of distinct elements

- For an integer $p > 0$, let $\text{zeros}(p)$ be the number of zeros at the end of the binary representation of p.

Counting the number of distinct elements

- For an integer $p > 0$, let zeros(p) be the number of zeros at the end of the binary representation of p.

$$\text{zeros}(p) = \max\{i \mid 2^i \text{ divides } p\}.$$

Counting the number of distinct elements

- For an integer $p > 0$, let zeros(p) be the number of zeros at the end of the binary representation of p.

$$\text{zeros}(p) = \max\{i \mid 2^i \text{ divides } p\}.$$

- Algorithm:

Count-Dif(stream s)
Choose a random hash function $h : [n] \to [n]$ form a universal family
int $z = 0$
while not $s.end()$ **do**
    $j = s.read()$
    if zeros($h(j)$) $> z$ **then**
        $z =$ zeros($h(j)$)
    end if
end while
Return $2^{z+\frac{1}{2}}$

Counting the number of distinct elements

- For an integer $p > 0$, let zeros(p) be the number of zeros at the end of the binary representation of p.

$$\text{zeros}(p) = \max\{i \mid 2^i \text{ divides } p\}.$$

- Algorithm:

```
Count-Dif(stream s)
Choose a random hash function h : [n] → [n] form a universal
family
int z = 0
while not s.end() do
    j = s.read()
    if zeros(h(j)) > z then
        z = zeros(h(j))
    end if
end while
Return 2^(z+1/2)
```

- Assuming that there are d distinct elements, the algorithm computes $\max \text{zeros}(h(j))$ as a good approximation of $\log d$.

Counting the number of distinct elements: Quality

- 1 pass, $O(\log n + \log\log n)$ memory and $O(1)$ time per item.

Counting the number of distinct elements: Quality

- 1 pass, $O(\log n + \log\log n)$ memory and $O(1)$ time per item.
- For $j \in [n]$ and $r \geq 0$, let $X_{r,j}$ be the indicator r.v. for
  $$\text{zeros}(h(j)) \geq r.$$
- Let $Y_r = \sum_{j \mid f_j > 0} X_{r,j}$.
- Let t denote the final value of z.

Counting the number of distinct elements: Quality

- 1 pass, $O(\log n + \log\log n)$ memory and $O(1)$ time per item.
- For $j \in [n]$ and $r \geq 0$, let $X_{r,j}$ be the indicator r.v. for
  $\text{zeros}(h(j)) \geq r$.
- Let $Y_r = \sum_{j|f_j>0} X_{r,j}$.
- Let t denote the final value of z.
- $Y_r > 0$ iff $t \geq r$, or equivalently $Y_r = 0$ iff $t \leq r - 1$.

Counting the number of distinct elements: Quality

- 1 pass, $O(\log n + \log\log n)$ memory and $O(1)$ time per item.
- For $j \in [n]$ and $r \geq 0$, let $X_{r,j}$ be the indicator r.v. for $\text{zeros}(h(j)) \geq r$.
- Let $Y_r = \sum_{j|f_j>0} X_{r,j}$.
- Let t denote the final value of z.
- $Y_r > 0$ iff $t \geq r$, or equivalently $Y_r = 0$ iff $t \leq r - 1$.
- Since $h(j)$ is uniformly distributed over the $\log n$-bit strings,

$$E[X_{r,j}] = Pr[\text{zeros}(h(j)) \geq r] = Pr[2^r \text{ divides } h(j)] = \frac{1}{2^r}$$

Counting the number of distinct elements: Quality

$$E[X_{r,j}] = Pr[\text{zeros}(h(j)) \geq r] = Pr[2^r \text{ divides } h(j)] = \frac{1}{2^r}.$$

Counting the number of distinct elements: Quality

$$E[X_{r,j}] = Pr[\text{zeros}(h(j)) \geq r] = Pr[2^r \text{ divides } h(j)] = \frac{1}{2^r}.$$

$$E[Y_r] = \sum_{j|f_j>0} E[X_{r,j}] = \frac{d}{2^r}$$

Counting the number of distinct elements: Quality

$$E[X_{r,j}] = Pr[\text{zeros}(h(j)) \geq r] = Pr[2^r \text{ divides } h(j)] = \frac{1}{2^r}.$$

$$E[Y_r] = \sum_{j|f_j>0} E[X_{r,j}] = \frac{d}{2^r}$$

- Random variables Y_r are pairwise independent, as they come from a universal hash family.

$$Var[Y_r] = \sum_{j|f_j>0} Var[X_{r,j}] \leq \sum_{j|f_j>0} E[X_{r,j}^2] = \sum_{j|f_j>0} E[X_{r,j}] = \frac{d}{2^r}$$

Counting the number of distinct elements: Quality

- $E[Y_r] = Var[Y_r] = d/2^r$
- Using Markov's and Chebyshev's inequalities,

$$Pr[Y_r > 0] = Pr[Y_r \geq 1] \leq \frac{E[Y_r]}{1} = \frac{d}{2^r}.$$

$$Pr[Y_r = 0] = Pr[|Y_r - E[Y_r]| \geq \frac{d}{2^r}] \leq \frac{Var[Y_r]}{(d/2^r)^2} \leq \frac{2^r}{d}.$$

Counting the number of distinct elements: Quality

- $Pr[Y_r > 0] \leq \frac{d}{2^r}$ and $Pr[Y_r = 0] \leq \frac{2^r}{d}$.

Counting the number of distinct elements: Quality

- $Pr[Y_r > 0] \leq \frac{d}{2^r}$ and $Pr[Y_r = 0] \leq \frac{2^r}{d}$.
- Let $\hat{d}$ be the estimate of d, $\hat{d} = 2^{t+\frac{1}{2}}$.

Counting the number of distinct elements: Quality

- $Pr[Y_r > 0] \leq \frac{d}{2^r}$ and $Pr[Y_r = 0] \leq \frac{2^r}{d}$.
- Let $\hat{d}$ be the estimate of d, $\hat{d} = 2^{t+\frac{1}{2}}$.
- Let a be the smallest integer so that $2^{a+\frac{1}{2}} \geq 3d$,

$$Pr[\hat{d} \geq 3d] = Pr[t \geq a] = Pr[Y_a = 0] \leq \frac{d}{2^a} \leq \frac{\sqrt{2}}{3}.$$

Counting the number of distinct elements: Quality

- $Pr[Y_r > 0] \leq \frac{d}{2^r}$ and $Pr[Y_r = 0] \leq \frac{2^r}{d}$.
- Let $\hat{d}$ be the estimate of d, $\hat{d} = 2^{t+\frac{1}{2}}$.
- Let a be the smallest integer so that $2^{a+\frac{1}{2}} \geq 3d$,

$$Pr[\hat{d} \geq 3d] = Pr[t \geq a] = Pr[Y_a = 0] \leq \frac{d}{2^a} \leq \frac{\sqrt{2}}{3}.$$

- Let b be the largest integer so that $2^{b+\frac{1}{2}} \leq 3d$,

$$Pr[\hat{d} \leq 3d] = Pr[t \leq b] = Pr[Y_{b+1} = 0] \leq \frac{2^{b+1}}{d} \leq \frac{\sqrt{2}}{3}.$$

Counting the number of distinct elements: Quality

- $Pr[\hat{d} \geq 3d] \leq \frac{\sqrt{2}}{3}$ and $Pr[\hat{d} \leq 3d] \leq \frac{\sqrt{2}}{3}$.
- Thus the algorithm provides a $(2, \frac{\sqrt{2}}{3})$-approximation.

Counting the number of distinct elements: Quality

- $Pr[\hat{d} \geq 3d] \leq \frac{\sqrt{2}}{3}$ and $Pr[\hat{d} \leq 3d] \leq \frac{\sqrt{2}}{3}$.
- Thus the algorithm provides a $(2, \frac{\sqrt{2}}{3})$-approximation.
- How to improve the quality of the approximation?

Counting the number of distinct elements: Quality

- $Pr[\hat{d} \geq 3d] \leq \frac{\sqrt{2}}{3}$ and $Pr[\hat{d} \leq 3d] \leq \frac{\sqrt{2}}{3}$.
- Thus the algorithm provides a $(2, \frac{\sqrt{2}}{3})$-approximation.
- How to improve the quality of the approximation?
- Usual technique: run k several independent copies of the algorithm and take the best information from them, in this case,

Counting the number of distinct elements: Quality

- $Pr[\hat{d} \geq 3d] \leq \frac{\sqrt{2}}{3}$ and $Pr[\hat{d} \leq 3d] \leq \frac{\sqrt{2}}{3}$.
- Thus the algorithm provides a $(2, \frac{\sqrt{2}}{3})$-approximation.
- How to improve the quality of the approximation?
- Usual technique: run k several independent copies of the algorithm and take the best information from them, in this case, the median of the k answers.

Counting the number of distinct elements: Quality

- $Pr[\hat{d} \geq 3d] \leq \frac{\sqrt{2}}{3}$ and $Pr[\hat{d} \leq 3d] \leq \frac{\sqrt{2}}{3}$.
- Thus the algorithm provides a $(2, \frac{\sqrt{2}}{3})$-approximation.
- How to improve the quality of the approximation?
- Usual technique: run k several independent copies of the algorithm and take the best information from them, in this case, the median of the k answers.

  If the median exceed $3d$ at least $k/2$ of the runs do.

Counting the number of distinct elements: Quality

- $Pr[\hat{d} \geq 3d] \leq \frac{\sqrt{2}}{3}$ and $Pr[\hat{d} \leq 3d] \leq \frac{\sqrt{2}}{3}$.
- Thus the algorithm provides a $(2, \frac{\sqrt{2}}{3})$-approximation.
- How to improve the quality of the approximation?
- Usual technique: run k several independent copies of the algorithm and take the best information from them, in this case, the median of the k answers.

  If the median exceed $3d$ at least $k/2$ of the runs do.
- By standard Chernoff bounds, the median exceed $3d$ with probability $2^{-\Omega(k)}$ and the median is below $3d$ with probability $2^{-\Omega(k)}$.

Counting the number of distinct elements: Quality

- $Pr[\hat{d} \geq 3d] \leq \frac{\sqrt{2}}{3}$ and $Pr[\hat{d} \leq 3d] \leq \frac{\sqrt{2}}{3}$.
- Thus the algorithm provides a $(2, \frac{\sqrt{2}}{3})$-approximation.
- How to improve the quality of the approximation?
- Usual technique: run k several independent copies of the algorithm and take the best information from them, in this case, the median of the k answers.

  If the median exceed $3d$ at least $k/2$ of the runs do.
- By standard Chernoff bounds, the median exceed $3d$ with probability $2^{-\Omega(k)}$ and the median is below $3d$ with probability $2^{-\Omega(k)}$.
- Choosing $k = \Theta(\log(1/\delta))$, we can make the sum to be at most δ. So we get a $(2, \delta)$-approximation. However, the used memory is now $O(log(1/\delta) \log n)$.