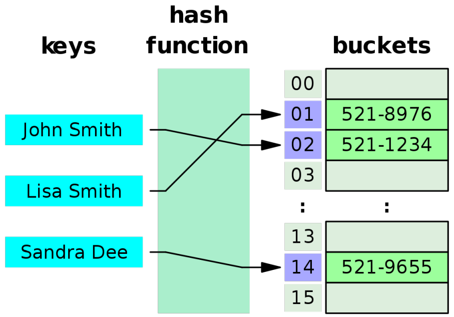


Perfect hashing
Lecture 11

COSC 242 – Algorithms and Data Structures

Today’s outline

1. Hashing review
2. Probabilities
3. Perfect hashing
4. Perfect hashing example

Today’s outline

1. Hashing review
2. Probabilities
3. Perfect hashing
4. Perfect hashing example

Hashing review

The point of hashing is:

- To get fast insert and search. Ideally, O(1) on average.
- To use storage space much smaller than the space of all possible key values, closer to the optimum of just enough space for the data.

A hash table is essentially an array.

Put the key *k* into position *h*(*k*) if that cell is empty.

If cell *h*(*k*) is not empty (a collision) either put it:

- Open addressing: somewhere else, or
- Chaining: Into a linked list at position *h*(*k*)

Subtleties

For **open addressing**:

- Need a collision resolution strategy.
- Deletion: need lazy deletion to not disrupt search path (but table fills up).
- Rehashing required for full tables, which is costly, so suboptimal for data sets that grow and shrink.

Subtleties

For **chaining**:

- Insertion: (-) overhead of maintaining list, (+) still O(1) to insert.
- Retrieval: (-) *O*(*n*) search of list, (+) lists are short for a good hash function.
- Deletion: (-) *O*(*n*) search of list, plus list management overhead, (+) retrieval speed not impeded by real deletes.

Today’s outline

1. Hashing review
2. Probabilities
3. Perfect hashing
4. Perfect hashing example

Perfect hashing

Hashing is a good choice for its excellent average-case performance.

Hashing can also provide excellent *worst-case* performance when the set of keys is static: once the keys are stored in the table, the set of keys never changes.

If the set of keys is static (i.e. there will never be any insertions or deletions), we can design the hash function to get a worst-case search = O(1). This technique is called **perfect hashing**.

Perfect hashing

Example of static data: consider the set of file names on a CD-ROM. Or: the set of reserved words in a programming language.

Ideally with perfect hashing there are no collisions. If we can randomly generate a hash function that gives a collision infrequently, then we can generate new hash functions until there are no collisions.

Universal hashing with a big table

Universal hashing, on average, will produce a collision between two random keys 1/*m* of the time, for table size *m*.

What's the probability of at least one collision between *n* keys?

This value has a special name in mathematics, and it's called the **binomial coefficient**[1, 2]: $\binom{n}{k}$

This symbol is usually read as "n choose k", which means there are $\binom{n}{k}$ ways to choose an (unordered) subset of *k* elements from a fixed set of *n* elements.

Birthday paradox

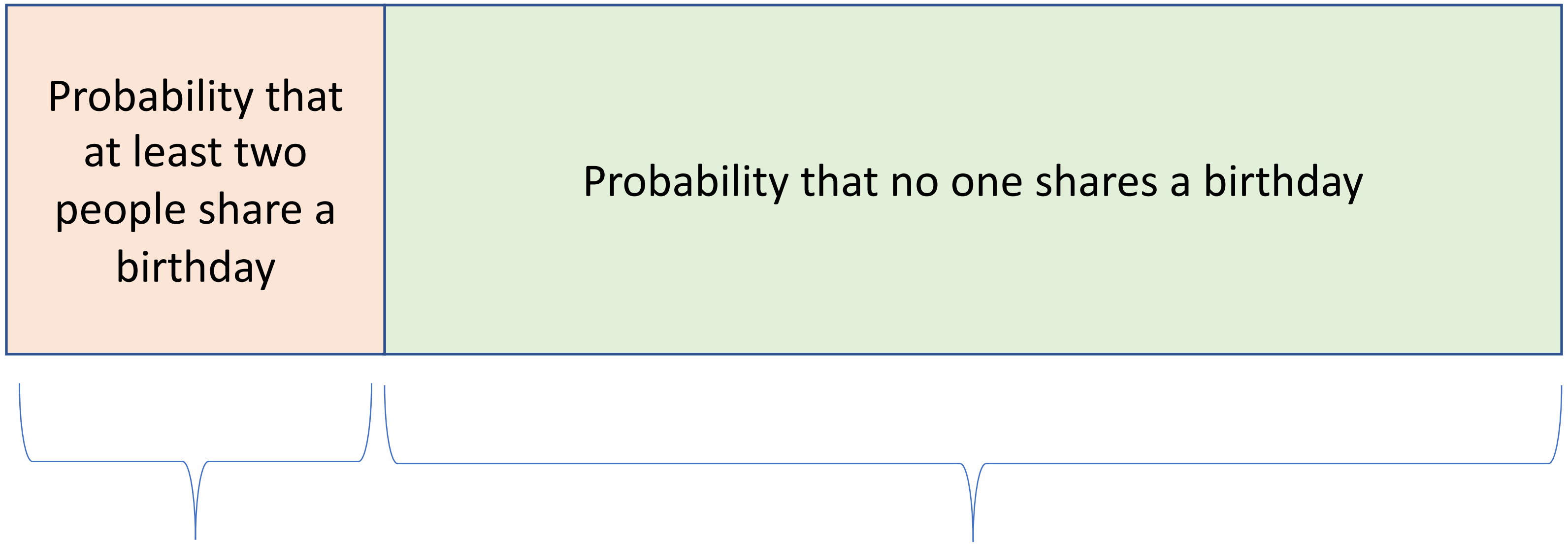
Our hashing collision problem is similar to a well known problem in math called the “birthday problem”[3].

In a room of *n* people, what’s the probability that at least one pair will have the same birthday?

It’s quite tricky to solve for this probability, which we’ll call P(A).

The key is a simplifying observation: The total probability (1) is equal to the probability of at least 2 people sharing a birthday, P(A), plus the probability that no one shares a birthday with anyone else, P(A’).

Visualizing probability



Probability that at least two people share a birthday

Probability that no one shares a birthday

1 = P(A) + P(A’)

Calculating P(A’)

This can be rewritten then as: P(A) = 1 - P(A’). We do this because calculating P(A’) is easier.

For the first person, the number of choices they have for a birthday not to collide with an existing birthday is: $\frac{365}{365}$. The next person now has 1 less dates to choose from: $\frac{364}{365}$. The next has $\frac{363}{365}$, and so on.

The conditional probability for P(A’) is:

$$P(A') = \frac{365}{365} \times \frac{364}{365} \times \frac{363}{365} \times \cdots \times \frac{365 - n + 1}{365}$$

Calculating P(A’)

Lets say we have 30 people in our room:

$$P(A') = \frac{365}{365} \times \frac{364}{365} \times \frac{363}{365} \times \ldots \times \frac{336}{365} = \frac{365 \cdot 364 \cdot \cdots \cdot 336}{(365)^{30}}$$

Can we rewrite our numerator as a factorial?

Calculating P(A’)

Lets say we have 30 people in our room:

$$P(A') = \frac{365}{365} \times \frac{364}{365} \times \frac{363}{365} \times \ldots \times \frac{336}{365} = \frac{365 \cdot 364 \cdots \cdot 336}{(365)^{30}}$$

Can we rewrite our numerator as a factorial?

$$365! = \boxed{365 \times 364 \times \cdots \times 336} \times \boxed{335 \times 334 \times \cdots \times 1}$$

Calculating P(A’)

Lets say we have 30 people in our room:

$$P(A') = \frac{365}{365} \times \frac{364}{365} \times \frac{363}{365} \times \ldots \times \frac{336}{365} = \frac{365 \cdot 364 \cdots 336}{(365)^{30}}$$

Can we rewrite our numerator as a factorial?

$$365! = \boxed{365 \times 364 \times \cdots \times 336} \times \boxed{335 \times 334 \times \cdots \times 1}$$

Where:

$$\frac{365!}{\boxed{335 \times 334 \times \cdots \times 1}} = \boxed{365 \times 364 \times \cdots \times 336} = \frac{365!}{(365 - 30)!}$$

Calculating P(A’)

Lets say we have 30 people in our room:

$$P(A') = \frac{365}{365} \times \frac{364}{365} \times \frac{363}{365} \times \ldots \times \frac{336}{365} = \frac{365 \cdot 364 \cdot \cdots \cdot 336}{(365)^{30}}$$

Can we rewrite our numerator as a factorial?

$$365! = \boxed{365 \times 364 \times \cdots \times 336} \times \boxed{335 \times 334 \times \cdots \times 1}$$

Where:

$$\frac{365!}{\boxed{335 \times 334 \times \cdots \times 1}} = \boxed{365 \times 364 \times \cdots \times 336} = \frac{365!}{(365-30)!}$$

$$P(A') = \left. \frac{365!}{(365-30)!} \middle/ (365)^{30} \right. = 29.37\%$$

Therefore, P(A) = 1-.2937 = 0.706

Calculating collisions

Lets return to our hash table. With a table size *m*, and *n* keys. What's the probability of at least one collision between *n* keys?

To calculate this, first need to calculate the number of possible key pairs.

We already showed how to do this with the numerator in our birthday paradox: $\frac{n!}{(n-2)!}$.

This formula is for probabilities where the order mattered. In our hashing example, we don't care about the order. To handle this, we further divide by n!

How many pairs?

Why divide by n!? Think of it this way: How many triplets of the numbers 1, 2, 3 can you form?

You can make 6 = ([1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2 ,1]), or 3!

These triplets still involve the same three numbers. For our hashing example, we don’t care about uniqueness (order).

We only want to count the set of numbers (1, 2, 3) in any order once. So we divide by *k*! [4]

How many pairs?

The number of possible key pairs is:

$$\binom{n}{k} = \frac{n!}{(n-2)!\,2!}$$

where $\binom{n}{k}$ is our binomial coefficient (*n* choose *k*)

$$= \frac{n(n-1)(n-2)\ldots 1}{(n-2)(n-3)\cdot\cdots\cdot 1\cdot 2\cdot 1}$$

$$= \frac{n(n-1)}{2}$$

Universal hashing with a big table

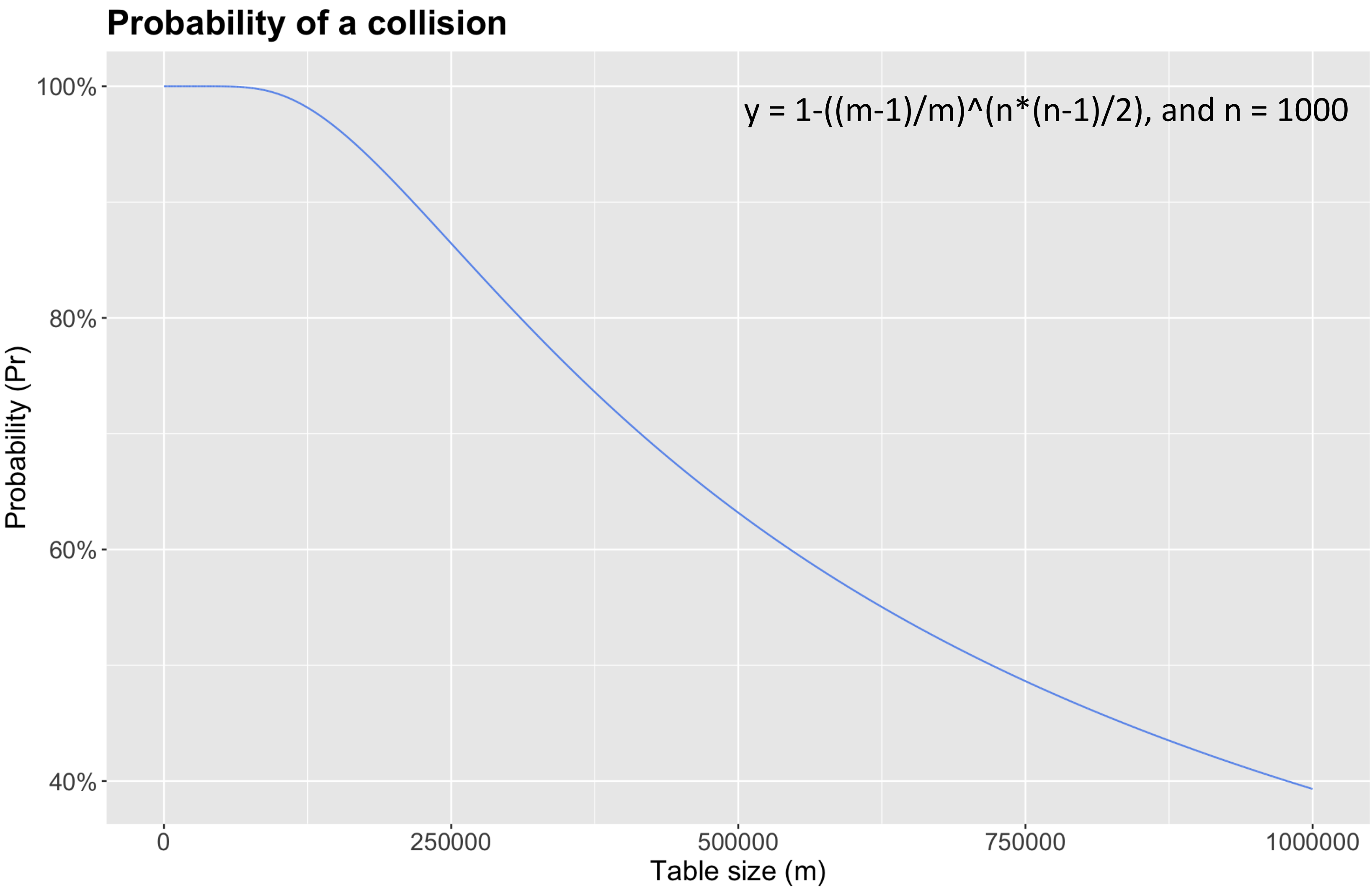
Each pair has a probability of collision = ${}^{1}/_{m}$. The probability of at least one collision for a random universal hash function is:

$$
\begin{aligned}
\Pr(\#\text{collisions} \geq 1) &= 1 - \Pr(\#\text{collisions} = 0) \\
&= 1 - \left(1 - \frac{1}{m}\right)^{\left({}^{n(n-1)}/_{2}\right)} = 1 - \boxed{\left(1 - \frac{1}{m}\right)}^{\binom{n}{k}} \\
&= 1 - \left(\frac{m-1}{m}\right)^{\left({}^{n(n-1)}/_{2}\right)}
\end{aligned}
$$

Prob of one pair(#=0) (#number of pairs)

How big should we make *m* if we want the probability of a collision to be low?

Universal hashing with a big table



Probability of a collision

y = 1-((m-1)/m)^(n*(n-1)/2), and n = 1000

Choosing m

With n = 1000, m = 100000, then Pr((#collisions ≥ 1) = .9932

If we choose 100 random functions, what's the probability that *all* of them have a collision?

$$.9932^{100} = .507$$

If we choose 100 random functions, then there is a 50% chance that one of them has *no collisions*. We would need to choose 340 before the probability dropped below 10%.

And that's for an occupancy rate of 1% - not a great way of choosing a perfect hashing function.

Today’s outline

1. Hashing review
2. Probabilities
3. Perfect hashing
4. Perfect hashing example

Hash of hashes

To create a perfect hashing scheme, we use two levels of hashing, with universal hashing at each level.

Instead of making a linked list of the keys hashing to slot *j*, however, we use a small **secondary hash table** S_j with an associated hash function h_j.

By trying several hash functions from a universal class, we can easily achieve the goal of having no collisions in the secondary tables.

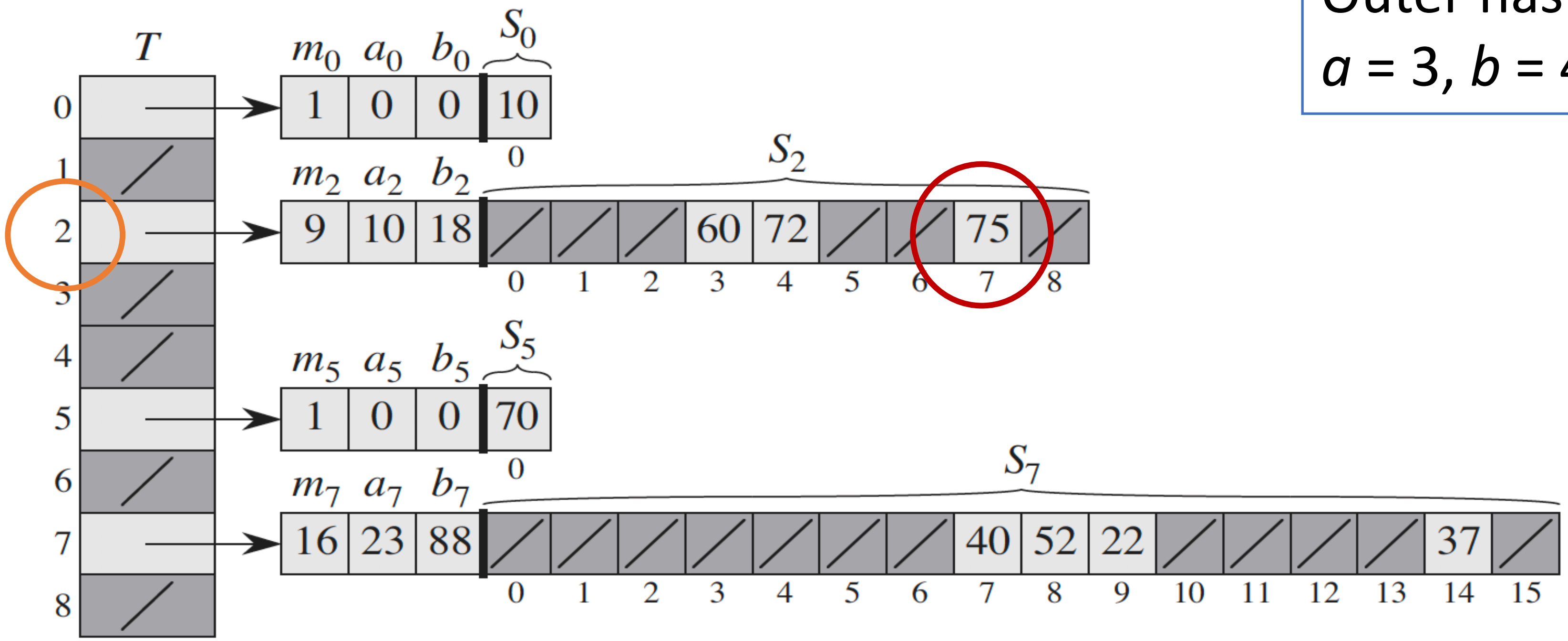
Hash of hashes

A similar “try out” principle on the primary hash function will allow us to choose a primary hash function that doesn't have lots of collisions, so that our table size for the primary hash table can be $m = O(n)$.

Recall that a member of a universal class of hash functions is fully specified by p, m, a, b where p is a prime greater than all possible keys, m is the size of the hash table, and a and b are chosen randomly from the range 1 up to p - 1.

Hash of hashes

K = {10, 22, 37, 40, 52, 60, 70, 72, 75}
Outer hash h(k) = (($ak + b$) % p) % m
a = 3, b = 42, p = 101, and m = 9



Example: h(75) = 2, and so key 75 hashes to slot 2 of table T. A secondary hash table S_j stores all keys hashing to slot j. The size of hash table S_j is $m_j = n_j^2$, and the associated hash function is $h_j(k) = \left((a_j k + b_j)\%p\right)\%m_j$

Since $h_2(75)$ = 7, key 75 is stored in slot 7 of secondary hash table S_2.

Perfect hashing

Main hash table: choose size *m* = *n* where *n* is the number of data items, choose prime *p* > the biggest key value. Choose *a* and *b* randomly to get primary hash function h(k) = ((ak + b)%p)%m.

Test *h* as follows:

1. For each key *k*, work out the home cell *h*(*k*). Keep a count n_i for each cell *i* of how many keys hash to *i*.
2. Check whether space required is too big: is the sum of all the n_i^2 giving a total > 2*n*? Then *h* is not good enough so repeat the process with new *a*, *b* for a new primary hash function.

Perfect hashing

If the primary hash function *h* is good enough:

1. For each slot *i*, get the secondary hash function h_i by setting $p_i = p$ (i.e. p doesn't change), setting $m_i = n_i^2$, and choosing a_i and b_i randomly.
2. Check to make sure that the resulting h_i doesn't cause any collisions within the secondary table.

Today’s outline

1. Hashing review
2. Probabilities
3. Perfect hashing
4. Perfect hashing example

Perfect hashing example

K = {8, 22, 36, 75, 61, 13, 84, 58}

$$h(k) = \big((ak+b)\%p\big)\%\,m$$
$$h_j(k) = \Big(\big(a_j k + b_j\big)\%p\Big)\,\%m$$

0	
1	
2	
3	
4	
5	
6	
7	

Step 1: Randomly generate values for *a* and *b*, select *p* > *K*
Step 2: For each key *k*, work out home cell *h*(*k*). Keep a count.
Step 3: Check if $\sum_{j=0}^{m-1} n_j^2 < 2n$
Step 4: Populate sub-tables, calculating new hash functions

Perfect hashing example

$h(k) = \big((ak + b)\%p\big)\%\, m$

$h_j(k) = \Big(\big(a_j k + b_j\big)\%p\Big)\ \%m$

K = {8, 22, 36, 75, 61, 13, 84, 58}

p = 87, *a* = 64, *b* = 5

0	
1	
2	
3	
4	
5	
6	
7	

Step 1: Randomly generate values for *a* and *b*, select *p* > *K*

Step 2: For each key *k*, work out home cell *h*(*k*). Keep a count.

Step 3: Check if $\sum_{j=0}^{m-1} n_j^2 < 2n$

Step 4: Populate sub-tables, calculating new hash functions

Perfect hashing example

$h(k) = \big((ak + b)\%p\big)\%\,m$

$h_j(k) = \Big(\big(a_j k + b_j\big)\%p\Big)\,\%m$

K = {8, 22, 36, 75, 61, 13, 84, 58}

p = 87, *a* = 64, *b* = 5

0	
1	
2	
3	
4	
5	
6	
7	

Step 1: Randomly generate values for *a* and *b*, select *p* > *K*

Step 2: For each key *k*, work out home cell *h*(*k*). Keep a count.

Step 3: Check if $\sum_{j=0}^{m-1} n_j^2 < 2n$

Step 4: Populate sub-tables, calculating new hash functions

Perfect hashing example

$$h(k) = \big((ak + b)\%p\big)\%\,m$$
$$h_j(k) = \Big(\big(a_jk + b_j\big)\%p\Big)\,\%m$$

K = {8, 22, 36, 75, 61, 13, 84, 58}

p = 87, *a* = 64, *b* = 5

0	/	0
1	/	61
2	/	8, 84
3	/	0
4	/	75
5	/	22
6	/	13
7	/	36, 58

Step 1: Randomly generate values for *a* and *b*, select *p* > *K*

Step 2: For each key *k*, work out home cell *h*(*k*). Keep a count.

Step 3: Check if $\sum_{j=0}^{m-1} n_j^2 < 2n$

Step 4: Populate sub-tables, calculating new hash functions

Perfect hashing example

$h(k) = \big((ak + b)\%p\big)\%\, m$

$h_j(k) = \Big(\big(a_j k + b_j\big)\%p\Big)\,\%m$

K = {8, 22, 36, 75, 61, 13, 84, 58}

p = 87, *a* = 64, *b* = 5

Step 1: Randomly generate values for *a* and *b*, select *p* > *K*

Step 2: For each key *k*, work out home cell *h*(*k*). Keep a count.

Step 3: Check if $\sum_{j=0}^{m-1} n_j^2 < 2n$

Step 4: Populate sub-tables, calculating new hash functions

			Count
0	/	0	0
1	/	61	1
2	/	8, 84	2
3	/	0	0
4	/	75	1
5	/	22	1
6	/	13	1
7	/	36, 58	2

Perfect hashing example

$h(k) = \big((ak + b)\%p\big)\%\, m$

$h_j(k) = \Big(\big(a_j k + b_j\big)\%p\Big)\, \%m$

K = {8, 22, 36, 75, 61, 13, 84, 58}

p = 87, *a* = 64, *b* = 5

			Count
0	/	0	0
1	/	61	1
2	/	8, 84	2
3	/	0	0
4	/	75	1
5	/	22	1
6	/	13	1
7	/	36, 58	2

Step 1: Randomly generate values for *a* and *b*, select *p* > *K*

Step 2: For each key *k*, work out home cell *h*(*k*). Keep a count.

Step 3: Check if $\sum_{j=0}^{m-1} n_j^2 < 2n$

Step 4: Populate sub-tables, calculating new hash functions

$$\sum_{j=0}^{m-1} n_j^2 = 0 + 1 + 4 + 0 + 1 + 1 + 1 + 4 = 12$$

$$12 < 2 * n = 16$$ ✓

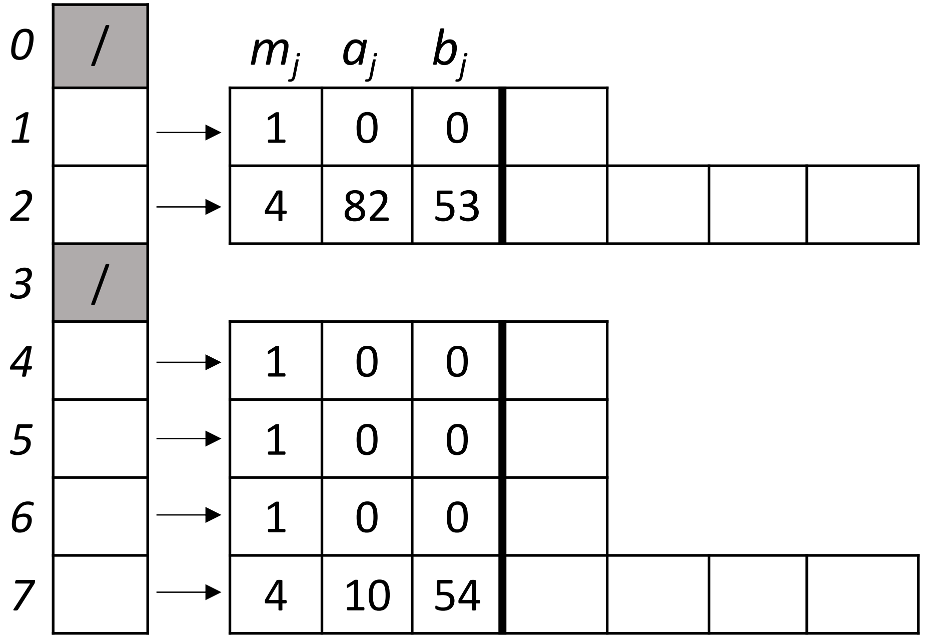
Perfect hashing example

$h(k) = \big((ak + b)\%p\big)\%\, m$

$h_j(k) = \Big(\big(a_j k + b_j\big)\%p\Big)\,\%m$

K = {8, 22, 36, 75, 61, 13, 84, 58}

p = 87, *a* = 64, *b* = 5



		m_j	a_j	b_j
0	/			
1		1	0	0
2		4	82	53
3	/			
4		1	0	0
5		1	0	0
6		1	0	0
7		4	10	54

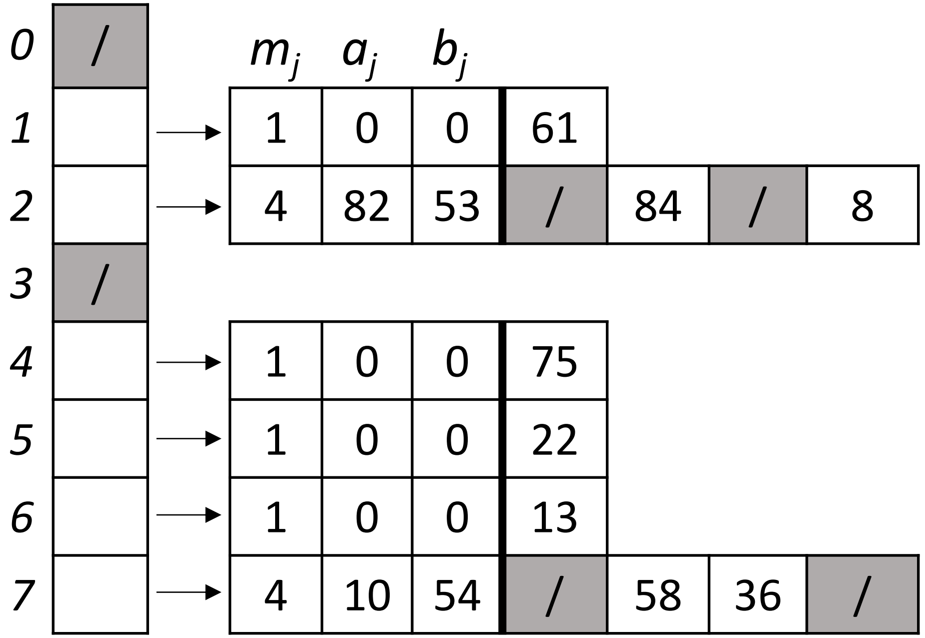
Step 4: Populate sub-tables, calculating new hash functions

Perfect hashing example

K = {8, 22, 36, 75, 61, 13, 84, 58}

p = 87, *a* = 64, *b* = 5

$$h(k) = \big((ak + b)\%p\big)\%\, m$$
$$h_j(k) = \Big(\big(a_j k + b_j\big)\%p\Big)\,\%m$$



		m_j	a_j	b_j				
0	/							
1		1	0	0	61			
2		4	82	53	/	84	/	8
3	/							
4		1	0	0	75			
5		1	0	0	22			
6		1	0	0	13			
7		4	10	54	/	58	36	/

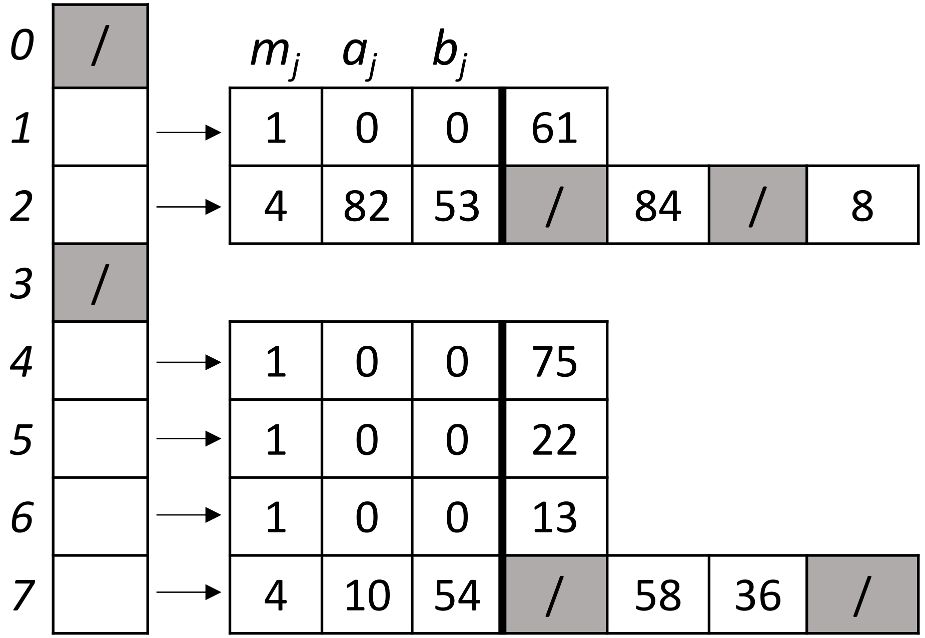
<u>Step 4</u>: Populate sub-tables, calculating new hash functions

Perfect hashing example

$$h(k) = \big((ak + b)\%p\big)\%\, m$$
$$h_j(k) = \Big(\big(a_j k + b_j\big)\%p\Big)\,\%m$$

K = {8, 22, 36, 75, 61, 13, 84, 58}

p = 87, *a* = 64, *b* = 5



		m_j	a_j	b_j				
0	/							
1		1	0	0	61			
2		4	82	53	/	84	/	8
3	/							
4		1	0	0	75			
5		1	0	0	22			
6		1	0	0	13			
7		4	10	54	/	58	36	/

No S_j keys hash to the same sub-slots. **All done!**

References

1. Binomial coefficient factorial formula - https://en.wikipedia.org/wiki/Binomial_coefficient#Factorial_formula
2. Binomial coefficient derivation - Kahn
3. Birthday paradox - Kahn
4. Derivation of key pair formula - https://www.mathsisfun.com/combinatorics/combinations-permutations.html

Suggested reading

Perfect hashing is discussed in Section 11.5.

Theorem 11.9 uses a different analysis of the probability of a collision to show that you need m = n^2 for a probability of collision less than 0.5.

The analysis in the textbook looks simpler, but there are some deeper concepts used.

Image attributions

This Photo by Unknown Author is licensed under CC BY

Disclaimer: Images and attribution text provided by PowerPoint search. The author has no connection with, nor endorses, the attributed parties and/or websites listed above.