COMP26120 Algorithms and Data Structures
Topic 2: Data Structures

Hash Functions

Dr. Thomas Carroll
thomas.carroll@manchester.ac.uk
All information on Blackboard

Learning Outcomes

- Understand the purpose of a Hash Function
- Appreciate what a “good” Hash Function might look like
- Know of different types of Hash Function

Where should we go?

0	
1	
2	
3	

Name	Number	%4
Unicorn	21	**1**
Dog	4	**0**
Tiger	20	**0**
Elephant	5	**1**
Octopus	15	**3**

A Few Issues...

- Pigeonhole Principle:
  - N buckets, M items; N < M; One bucket has to have more than one item.
- Birthday Paradox:
  - For *n* people chosen at random, the probability of two having the same birthday is larger than you might think!
  - for *n=23*, it's 50%; For *n=70*, its 90%!

Problems

Hashing Problem

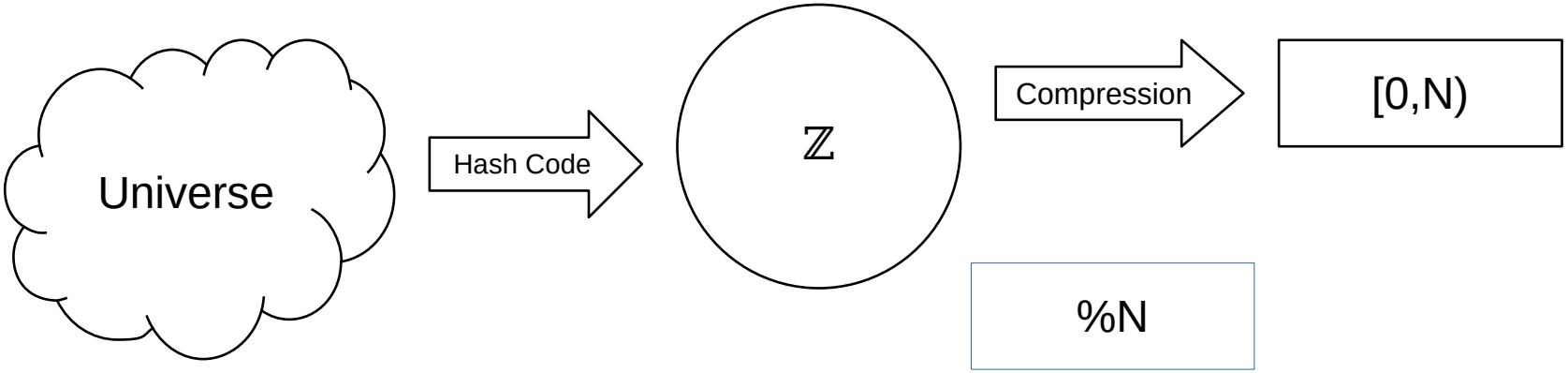
Given a Universe *U* of keys, create a function *hash* that maps each key to an integer in the range [0,N)

Uniform Hashing Problem

Given a Universe *U* of keys, create a function *hash* that **uniformly** maps each key to an integer in the range [0,N)
ie: each key is equally likely to map to each slot

What is a Hash Function?

- A Hash Function *maps* the universe to a set of integers 0 → N-1



Getting some Uniformity in Compression

- N could be **prime**
  - Say the hash code outputs something in the pattern: *a+bk % N*
  - Less likely for patterns to occur

Getting some Uniformity in Compression

- N could be a power of 2
  - $x \% 2^k$
  - Equivelant to bitwise:
    - $x \& (2^k-1)$
  - More collisions can occur due to indexing on least significant bits
  - We can do XOR with shifted key to mix in higher level bits:
    - x = x XOR x >>> 16

Implementing Your Own Compression

- You will probably want to use **prime-sized** hash tables
- You can explore power of 2, but you need to ensure some uniformity

Hash Codes

- The *thing* we want to hash can be seen as a sequence of integers:

  $a_0, a_1, a_2, \ldots, a_n$

- *We want to change the **sequence** of integers into a **single** integer*

$\sum a_i \quad \oplus a_i$

Symmetry is our enemy!

tab = bat
cat = tac

Avoiding Symmetry

- Polynomial based hash codes can help us avoid symmetry
- For some sequence $a_0, a_1, a_2, \ldots, a_{n-1}$, *some constant* c
- $a_0c^{n-1} + a_1c^{n-2} + \ldots + a_{n-1}$
- This creates “space” for each value in the sequence
  - First “gap” you map a0
  - Second “gap” you map a1, and so on…
- What value of **c?**
  - Java uses 31

Conclusion

- Hash Functions map the universe to a range of integers, using two parts:
  - Hash codes
  - Compression
- Symmetry will cause collisions, but we can avoid this by using polynomial hash codes or prime-sized hash tables.