# Hash Functions, and Where to Find Them

# What is a hash function?

## unbound data -> bound data

e.g. String -> UInt64

# What is it used for?

- data partitioning
- digital signature
- fingerprinting
- checksum

# Checksum

## Verify data integrity

# Fingerprinting

## Data identification

# Digital signature

## verifying the authenticity

# Data partitioning

## Grouping data

low collision (perfect hash function)

high collision (locality sensitive hashing, data clustering)

# Simplest data partitioning function

**A % N**

but **A** needs to be a positive number

# Why do we need a hash function in std lib?

## We want to have an associative array (dict)

But maybe we want more (see Zig std lib)

# Where do we find good hash functions?

## Or rather what is a good hash function?

# Good hash function

Fast

High quality

# What is high quality?

## Depends on the use case

# High quality for hash map

- It should ensure that two different keys are no more likely than random chance to produce the same hash value
- No matter what type of keys are used, a hash table built using the hash function should have its keys distributed randomly between the table cells.

https://github.com/aappleby/smhasher/wiki/Distribution

# SMHasher project

- https://github.com/aappleby/smhasher
- https://github.com/rurban/smhasher
- https://gitlab.com/fwojcik/smhasher3

# SMHasher info

SMHasher's test functions can be broken down into 4 classes -

1. Sanity. Does the hash function work? Does it do bad things with its input?
2. Performance. How fast can the hash function consume input, and how many cycles does it take to complete a hash?
3. Differentials. How often do almost-but-not-quite identical keys produce identical hash values? What is the smallest difference between two keys that can possibly cause the hash function to produce identical values?
4. Keysets. Take a bunch of keys, hash them, and analyze the resulting hash values. How evenly are the hash values spread over the 2^N-bit space (I've come up with an interesting metric for this which you can read about [FillFactor here]). How often do unrelated keys produce identical hash values, and does this occur more likely than would be expected due to statistics?

To address #4, we generate a variety of different hard-to-hash keysets and run our collision & [Distribution distribution] tests against all of them.

https://github.com/aappleby/smhasher/wiki/SMHasher

# SMhasher

Default timings with a modern AMD Ryzen 5 PRO 3350G 3.6GHz workstation:

| Hash function | MiB/sec | cycl./hash | cycl./map | size | Quality problems |
|---|---|---|---|---|---|
| donothing32 | 11149460.06 | 4.00 | - | 13 | test NOP |
| donothing64 | 11787676.42 | 4.00 | - | 13 | test NOP |
| donothing128 | 11745060.76 | 4.06 | - | 13 | test NOP |
| NOP_OAAT_read64 | 11372846.37 | 14.00 | - | 47 | test NOP |
| o1hash | 11629440.57 | 18.15 | 199.35 (2) | 101 | insecure, zeros, fails all tests |
| fletcher4 | 15556.93 | 20.60 | 358.60 (3) | 371 | UB, fails all tests |
| fletcher2 | 15552.61 | 20.61 | 335.31 (3) | 248 | UB, fails all tests |
| rapidhash | 23789.79 | 22.80 | 138.71 (7) | 574 | |
| fibonacci | 16878.32 | 22.94 | 803.18 (15) | 1692 | UB, zeros, fails all tests |
| sumhash32 | 42877.79 | 23.12 | - | 863 | UB, test FAIL |
| ahash64 | 14705.55 | 23.34 | 458.05 (14) | 412 | rust |
| rapidhash_unrolled | 23892.88 | 23.41 | 139.47 (12) | 782 | |
| multiply_shift | 8026.77 | 26.05 | 226.80 (8) | 345 | fails all tests |
| FNV1A_Totenschiff | 6274.78 | 26.23 | 251.13 (2) | 270 | UB, zeros, fails all tests |
| FNV1A_Pippip_Yurii | 6172.14 | 27.55 | 244.80 (2) | 147 | UB, sanity, fails all tests |
| wyhash | 22540.23 | 28.87 | 236.16 (8) | 474 | |
| wyhash32low | 22393.77 | 29.04 | 243.40 (3) | 474 | 5 bad seeds |
| aesni | 31232.34 | 29.21 | 230.14 (4) | 519 | machine-specific (x64 AES-NI) |
| xxh3 | 21033.55 | 29.48 | 226.77 (4) | 744 | Moment Chi2 14974, BIC |
| sumhash | 10699.57 | 29.53 | - | 363 | test FAIL |
| aesni-low | 31221.14 | 29.64 | 226.18 (3) | 519 | machine-specific (x64 AES-NI) |
| FNV1a_YT | 13486.49 | 30.50 | 237.43 (4) | 321 | UB, fails all tests |
| xxh3low | 17093.19 | 30.57 | 242.07 (7) | 756 | Moment Chi2 1.8e+9 ! |
| t1ha1_64le | 13442.64 | 31.41 | 219.58 (3) | 517 | Avalanche |

# https://github.com/mzaks/mojo-hash

- AHash (101 Lines)
- WyHash (98 lines)
- fnv1a 32/64bit (23 lines)
- fxhash 32/64bit (52 lines)
- o1hash (14 lines)
- MD5 (120 lines)
- std hash (DJBX33A, 271 lines most of which is doc)

```
alias fnv_64_prime = 0x100000001b3
alias fnv_64_offset_bassis = 0xcbf29ce484222325

@always_inline
fn fnv1a64(s: String) -> UInt64:
    var hash: UInt64 = fnv_64_offset_bassis
    var buffer = s.unsafe_ptr()
    for i in range(s.byte_length()):
        hash ^= buffer.load(i).cast[DType.uint64]()
        hash *= fnv_64_prime
    return hash
```

```
fn o1_hash(s: String) -> UInt64:
    var p = s.unsafe_ptr()
    var bytes = s.byte_length()
    if bytes >= 4:
        var first = p.bitcast[DType.uint32]()[0]
        var middle = p.offset((bytes >> 1) - 2).bitcast[DType.uint32]()[0]
        var last = p.offset(bytes - 4).bitcast[DType.uint32]()[0]
        return ((first + last) * middle).cast[DType.uint64]()
    if bytes:
        var tail = (p[0].cast[DType.uint64]() << 16)
            | (p[bytes >> 1].cast[DType.uint64]() << 8)
            | p[bytes - 1].cast[DType.uint64]()
        return tail * 0xa0761d6478bd642
    return 0
```

# Corpus 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque orci urna, pretium et porta ac, porttitor sit amet sem. Fusce sagittis lorem neque, vitae sollicitudin elit suscipit et. In interdum convallis nisl in ornare. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Aliquam erat volutpat. Morbi mollis iaculis lectus ac tincidunt. Fusce nisi lacus, semper eu dignissim et, malesuada non mi. Sed euismod urna vel elit faucibus, eu bibendum ante fringilla. Curabitur tempus in turpis at mattis. Aliquam erat volutpat. Donec maximus elementum felis, sit amet dignissim augue tincidunt blandit. Aliquam fermentum, est eu mollis.

Corpus 1

Word count 100 | unique word count 81 | min key size 2 | avg key size 5.6399999999999997 | max key size 12

AHash avg hash compute 22.5 | hash colision 1.0 | hash colision mod 512 1.1549295774647887

Wyhash avg hash compute 26.5 | hash colision 1.0 | hash colision mod 512 1.0123456790123457

fnv1a32 avg hash compute 17.5 | hash colision 1.0 | hash colision mod 512 1.12328767123287 68

fnv1a64 avg hash compute 23.0 | hash colision 1.0 | hash colision mod 512 1.0249999999999999

fxHash32 avg hash compute 19.0 | hash colision 1.0 | hash colision mod 512 1.2238805970149254

fxHash64 avg hash compute 12.0 | hash colision 1.0 | hash colision mod 512 1.1884057971014492

o1Hash avg hash compute 14.5 | hash colision 1.0 | hash colision mod 512 1.2615384615384615

MD5 avg hash compute 6345.5 | hash colision 1.0 | hash colision mod 512 1.1081081081081081

std_Hash64 avg hash compute 32.0 | hash colision 1.0 | hash colision mod 512 2.8275862068965516

# Corpus 2

But I must explain to you how all this mistaken idea of denouncing pleasure and praising pain was born and I will give you a complete account of the system, and expound the actual teachings of the great explorer of the truth, the master-builder of human happiness. No one rejects, dislikes, or avoids pleasure itself, because it is pleasure, but because those who do not know how to pursue pleasure rationally encounter consequences that are extremely painful. Nor again is there anyone who loves or pursues or desires to obtain pain of itself, because it is pain, but because occasionally circumstances occur in which toil and pain can procure him some great pleasure. To take a trivial example, which of us ever undertakes laborious physical ...

Corpus 2

Word count 999 | unique word count 203 | min key size 1 | avg key size 4.7957957957957955 | max key size 14

AHash avg hash compute 17.7177177177177719 | hash colision 1.0 | hash colision mod 512 1.2083333333333333

Wyhash avg hash compute 31.781781781781781 | hash colision 1.0 | hash colision mod 512 1.2011834319526626

fnv1a32 avg hash compute 18.2182182182182219 | hash colision 1.0 | hash colision mod 512 1.2848101265822784

fnv1a64 avg hash compute 18.8688688688688869 | hash colision 1.0 | hash colision mod 512 1.2011834319526626

fxHash32 avg hash compute 18.2682682682682269 | hash colision 1.0 | hash colision mod 512 1.3716216216216217

fxHash64 avg hash compute 22.3223223223223221 | hash colision 1.0 | hash colision mod 512 1.4195804195804196

o1Hash avg hash compute 16.9169169169169218 | hash colision 1.0 | hash colision mod 512 1.6639344262295082

MD5 avg hash compute 5993.1431431431431429 | hash colision 1.0 | hash colision mod 512 1.1871345029239766

std_Hash64 avg hash compute 31.2312312312312231 | hash colision 1.0 | hash colision mod 512 4.7209302325581399

# Corpus 7

AbortMultipartUpload CompleteMultipartUpload CopyObject
CreateBucket CreateMultipartUpload DeleteBucket
DeleteBucketAnalyticsConfiguration DeleteBucketCors
DeleteBucketEncryption DeleteBucketIntelligentTieringConfiguration
DeleteBucketInventoryConfiguration DeleteBucketLifecycle
DeleteBucketMetricsConfiguration DeleteBucketOwnershipControls
DeleteBucketPolicy DeleteBucketReplication DeleteBucketTagging
DeleteBucketWebsite DeleteObject DeleteObjects DeleteObjectTagging
DeletePublicAccessBlock GetBucketAccelerateConfiguration
GetBucketAcl GetBucketAnalyticsConfiguration GetBucketCors
GetBucketEncryption GetBucketIntelligentTieringConfiguration...

Word count 161 | unique word count 142 | min key size 8 | avg key size 22.167701863354036 | max key size 43
AHash avg hash compute 17.080745341614907 | hash colision 1.0 | hash colision mod 512 1.1259842519685039
Wyhash avg hash compute 38.509316770186338 | hash colision 1.0 | hash colision mod 512 1.1259842519685039
fnv1a32 avg hash compute 30.1242236024844474 | hash colision 1.0 | hash colision mod 512 1.153225806451613
fnv1a64 avg hash compute 30.4347826086956522 | hash colision 1.0 | hash colision mod 512 1.1626016260162602
fxHash32 avg hash compute 20.1863354037267709 | hash colision 1.0 | hash colision mod 512 1.1259842519685039
fxHash64 avg hash compute 19.5652173913043478 | hash colision 1.0 | hash colision mod 512 1.153225806451613
o1Hash avg hash compute 14.2857142857142856 | hash colision 1.0 | hash colision mod 512 1.3240740740740742
MD5 avg hash compute 6168.0124223602484888 | hash colision 1.0 | hash colision mod 512 1.1171875
std_Hash64 avg hash compute 33.540372670807453 | hash colision 1.0 | hash colision mod 512 2.1343283582089554

# Corpus 8

```
Path("/usr/share/dict/words").read_text().splitlines()
```

Corpus 8
Word count 235976 | unique word count 235975 | min key size 1 | avg key size 9.5683416957656711 | max key size 28
AHash avg hash compute 18.427015176684634 | hash colision 1.0 | hash colision mod 512 460.890625
Wyhash avg hash compute 27.843226994835632 | hash colision 1.0 | hash colision mod 512 460.890625
fnv1a32 avg hash compute 19.277384140760077 | hash colision 1.0000211890444164 | hash colision mod 512 460.890625
fnv1a64 avg hash compute 18.671390310879072 | hash colision 1.0 | hash colision mod 512 460.890625
fxHash32 avg hash compute 17.52861872958832 | hash colision 1.0006784950978729 | hash colision mod 512 460.890625
fxHash64 avg hash compute 17.466465516267192 | hash colision 1.0001101928374656 | hash colision mod 512 460.890625
o1Hash avg hash compute 16.736235888395431 | hash colision 1.002208480565371 | hash colision mod 512 463.60707269155205
std_Hash64 avg hash compute 27.490083737329218 | hash colision 1.0010860342779568 | hash colision mod 512 2088.283185840708

# Conclusion and Wish List

- Current std lib hash is not slow but has poor quality
- Would love to see Zigs approach in Mojo in the future
- UInt128 type or, `multipliedFullWidth(other: UInt64) -> (high: UInt64, low: UInt64)` on UInt64