Today:

- Continue with hashing algorithms

  hashing functions

  ↳ sum and modulo

  ↳ sum and multiplication

  ↳ perfect hash function

  table size

  ↳ collision resolution

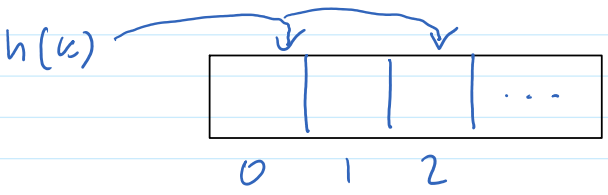
m = table size, k is key

$h(k) = sum(k) \% m$

Potential bias:

Example: data set with all sums result in even values

- h(k) will only generate even (or odd) values



Multiplication Method

1. Calculate sum of all ASCII values in key (k).
2. Multiply by a constant A,

where $0 < A < 1$

$k \cdot A \rightarrow$ A can be derived heuristically

Common value $A = 13/32$

3. keep the fractional part of kA

4. Scale kA fractional by table size (m)

5. Take the floor of the result to get integer index.

E.G. given $m = 1024$, $A = 13/32$, calculate hash value of string w/ 5 As and 5 Zs.

5 As

1) $k = 325$

2) $kA = 325\,(13/32) = 132.0312$

3) $fract = 0.0312$

4) $fract \cdot m = 0.0312 * 1024 = 32$

5 Zs

1)

2)

3)

4) $fract \cdot m = 0.8125 \times 1024 = 832$

Advantages of multiplication over sum modulo:

1) choosing m, less likely to cause a bias

2) multiplication is less expensive than %

Coding Q.

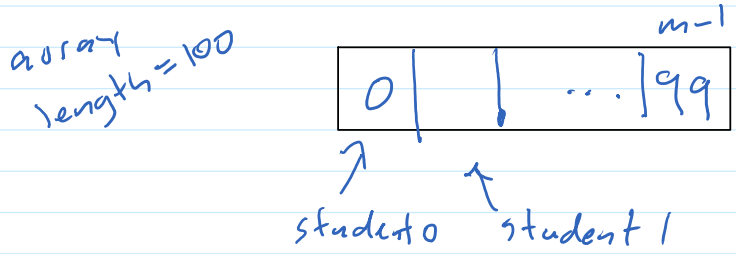
How to get fractional portion of a value? fract(132.0312) ⇒ 0.0312

```
fract(v)
    integer = floor(v)
    return(v - integer)
```

Say, want to store SID#s for 100 students, that happen to be sequential:

105608 900
901
902
⋮
999

$x = SID \% 100 \in Z \mid 0 \le x \le 99$



Perfect hash function all records assigned to the hash table w/o collisions or wasted space.

E.G. If SID #s not sequential, but more random.

$105608900 \% 100 = 0$

$105708900 \% 100 = 0$ } collision

What about the table size?

Larger table size = less collisions?

Example: Design a hashing algorithm for given data:

Possible key values:

- 2 letter strings
- can contain upper case letters from A to F

Design choices:

- hash function?
  - ↳ hash sum
- table size?
  - ↳

AA - FF

↑ min ↖ max

min = A + A = 65 + 65 = 130

max = F + F = 70 + 70 = 140

sum % m

	m=5	m=11	m=15
	%5	%11	%15
130	0	9	10
131	1	10	11
132	2	0	12
133	3	1	13
134	4	2	14
135	0	3	0
136	1	4	1
137	2	5	2
138	3	6	3
139	4	7	4
140	0	8	5

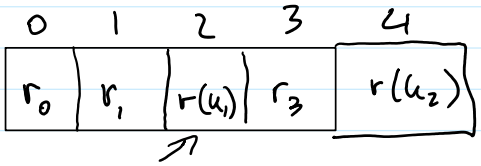
hashing function outputs

Takeaway: increasing table size above the possible range will not decrease # of collisions

Collision Resolution

One simple method: open addressing.

Add new record at next available location.



given some k_1 and k_2

$k_1 \neq k_2$, $h(k_1) = 2$

$h(k_2) = 2$

Another method: chaining

Every array element is a pointer to the head of a linked list.

given some k_1 and k_2 $h(k_1) = 2$

$h(k_2) = 2$