Data Structures and Algorithms
Hashing 3

CS 225
G Carl Evans

April 21, 2025

Department of Computer Science

Slides by Brad Solomon

A Hash Table based Dictionary

Client Code:

```
Dictionary<KeyType, ValueType> d;
d[k] = v;
```

A **Hash Table** consists of three things:

1. A hash function

2. A data storage structure

3. A method of addressing *hash collisions*

A Problem w/ Linear Probing

Primary clustering:

0	
1	1_1
2	1_2
3	3_1
4	1_3
5	3_2
6	
7	
8	
9	

Description:

Remedy:

(Example of closed hashing)

Collision Handling:

S = { 16, 8, 4, 13, 29, 11, 22 }

|S| = n

$h_1(k)$ = k % 7

|Array| = m

$h_2(k)$ = 1

$h(k, i) = (h_1(k) + i*h_2(k))$ % 7

Try $h(k) = (k + 0*h_2(k))$ % 7, if full…

Try $h(k) = (k + 1*h_2(k))$ % 7, if full…

Try $h(k) = (k + 2*h_2(k))$ % 7, if full…

Try …

0	
1	**8**
2	**16**
3	
4	**4**
5	
6	**13**

(Example of closed hashing)

Collision Handling: Double Hashing

S = { 16, 8, 4, 13, 29, 11, 22 } **|S| = n**

$h_1(k) = k \% 7$ **|Array| = m**

$h_2(k) = 5 - (k \% 5)$

0	
1	**8**
2	**16**
3	
4	**4**
5	
6	**13**

$h(k, i) = (h_1(k) + i*h_2(k)) \% 7$

Try $h(k) = (k + 0*h_2(k)) \% 7$, if full...

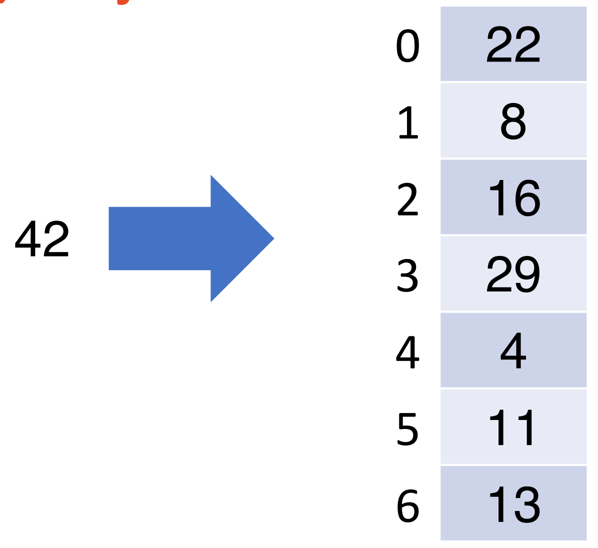
Try $h(k) = (k + 1*h_2(k)) \% 7$, if full...

Try $h(k) = (k + 2*h_2(k)) \% 7$, if full...

Try ...

What to do when the table is full?

S = { 16, 8, 4, 13, 29, 11, 22, 42 }

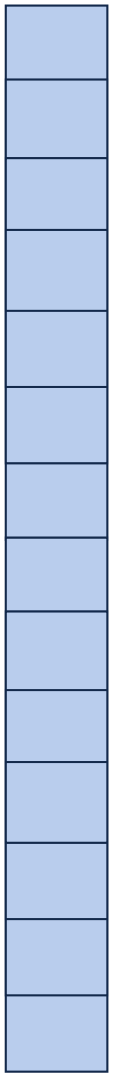


Resizing a hash table

How do we resize?

h(k, i) =

0	22
1	8
2	16
3	29
4	4
5	11
6	13



Running Times

	Hash Table	AVL	Linked List
Find			
Insert			
Storage Space			

Hash Function

Characteristics of a good hash function:

1. Computation Time:

2. Deterministic:

3. …

Simple Uniform Hashing Assumption

Given table of size m, a simple uniform hash, h, implies

$\forall k_1, k_2 \in U$ where $k_1 \neq k_2$, $Pr(h[k_1] = h[k_2]) = \frac{1}{m}$

Uniform:

Independent:

Separate Chaining Under SUHA

Given table of size m and n inserted objects

Claim: Under SUHA, expected length of chain is $\frac{n}{m}$

Running Times *(Don’t memorize these equations, no need.)*

(Expectation under SUHA)

Open Hashing:

insert: ___________.

find/ remove: ___________.

Closed Hashing:

insert: ___________.

find/ remove: ___________.

Running Times *(Don't memorize these equations, no need.)*

The expected number of probes for find(key) under SUHA

Linear Probing:

- Successful: **$\frac{1}{2}(1 + 1/(1-\alpha))$**
- Unsuccessful: **$\frac{1}{2}(1 + 1/(1-\alpha))^2$**

Double Hashing:

- Successful: **$1/\alpha * \ln(1/(1-\alpha))$**
- Unsuccessful: **$1/(1-\alpha)$**

Separate Chaining:

- Successful: **$1 + \alpha/2$**
- Unsuccessful: **$1 + \alpha$**

Instead, observe:

- **As α increases:**

- **If α is constant:**

Running Times

The expected number of probes for find(key) under SUHA

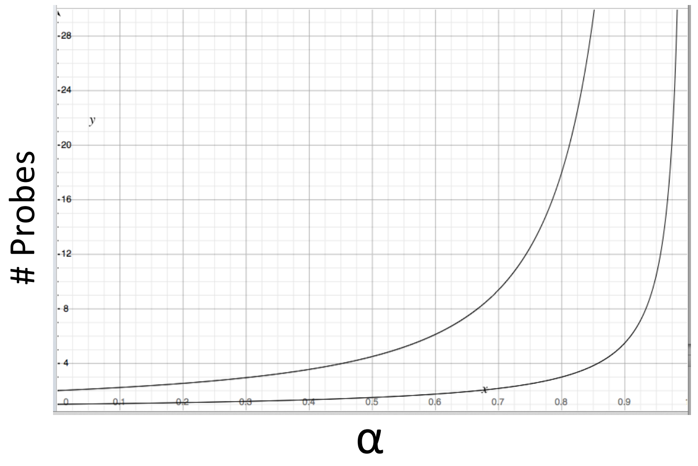
Linear Probing:

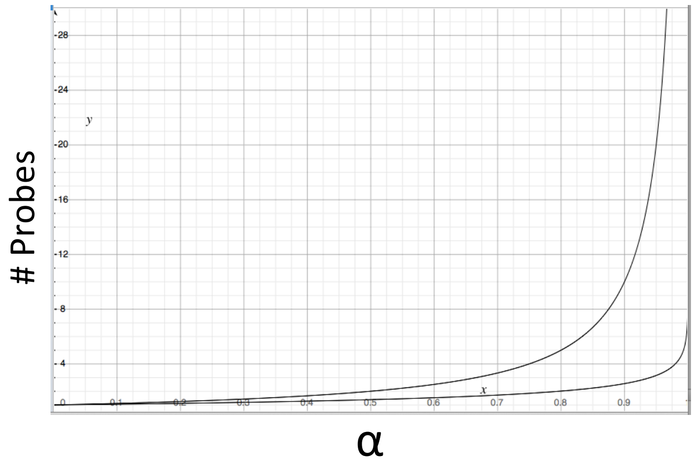
- Successful: **$\frac{1}{2}(1 + 1/(1-\alpha))$**
- Unsuccessful: **$\frac{1}{2}(1 + 1/(1-\alpha))^2$**

Double Hashing:

- Successful: **$1/\alpha * \ln(1/(1-\alpha))$**
- Unsuccessful: **$1/(1-\alpha)$**

When do we resize?





Which collision resolution strategy is better?

- Big Records:

- Structure Speed:

What structure do hash tables implement?

What constraint exists on hashing that doesn't exist with BSTs?

Why talk about BSTs at all?

Running Times

	Hash Table	AVL	Linked List
Find	Expectation*: Worst Case:		
Insert	Expectation*: Worst Case:		
Storage Space			

std data structures

std::map

::operator[]

::insert

::erase

::lower_bound(key) ➔ Iterator to first element ≤ key

::upper_bound(key) ➔ Iterator to first element > key

std data structures

std::unordered_map

::operator[]

::insert

::erase

~~::lower_bound(key) ➔ Iterator to first element ≤ key~~

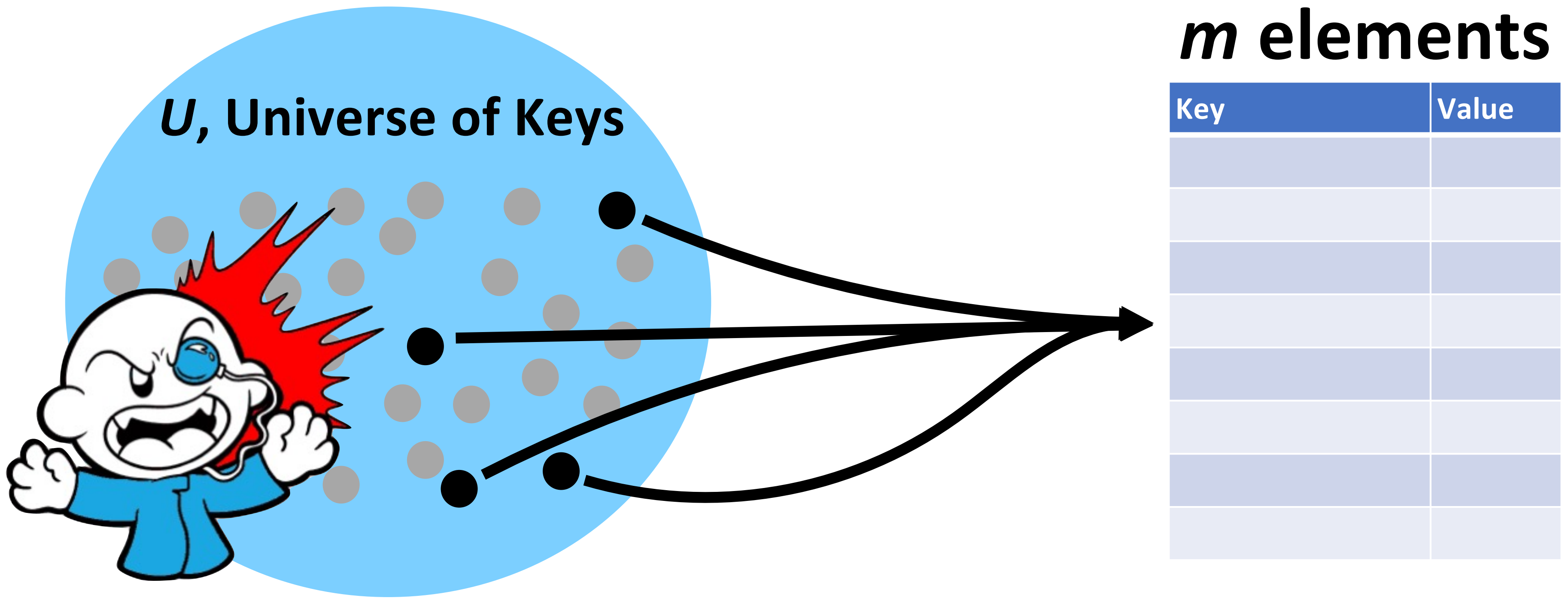
~~::upper_bound(key) ➔ Iterator to first element > key~~

::load_factor()

::max_load_factor(ml) ➔ Sets the max load factor

Hashing in the real world

Even under SUHA, our estimates are *in expectation*.



Hash Table

Worst-Case behavior is bad — but what about randomness?

1) **Fix *h***, our hash, and assume it is good for ***all keys***:

2) Create a ***universal hash function family:***

Hash Function (Division Method or Identity Hash)

Hash of form: $h(k) = k\%m$

Hash Function (Mid-Square Method)

Hash of form: $h(k) = (k * k)$ and take b middle bits where $m = 2^b$

Hash Function (Multiplication Method)

Hash of form: $h(k) = \lfloor m(remain(kA)) \rfloor,\ 0 \leq A \leq 1$

Hash Function (Universal Hash Family)

Pick a random $h \in H$ s.t. $\forall k_1, k_2 \in U,\ Pr(h[k_1] = h[k_2]) \leq \frac{1}{m}$

Hash Function (Universal Hash Family)

Hash of form: $h_{ab}(k) = ((ak + b)\%p)\%m,\ a, b \in Z_p^*, Z_p$

$\forall k_1 \neq k_2,\ Pr_{a,b}(h_{ab}[k_1] = h_{ab}[k_2]) \leq \frac{1}{m}$