Data Structures and Algorithms
Hashing 2

CS 225
Brad Solomon

November 13, 2024

Ooops no tablet -_-

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Learning Objectives

Review fundamentals of hash tables

Want an O(1) data structure! With probability we can get close in expectation!

Introduce **closed hashing** approaches to hash collisions

Determine when and how to resize a hash table

Justify when to use different index approaches

A Hash Table based Dictionary

User Code (is a map):

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

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

1. A hash function   Assigns numeric (positive int) address to any key

Key -> Hash Value (Address)

2. A data storage structure   Array — very good at lookup given **index**

Hash Value (Address) is an index!

3. A method of addressing *hash collisions*

Two different keys, same hash value

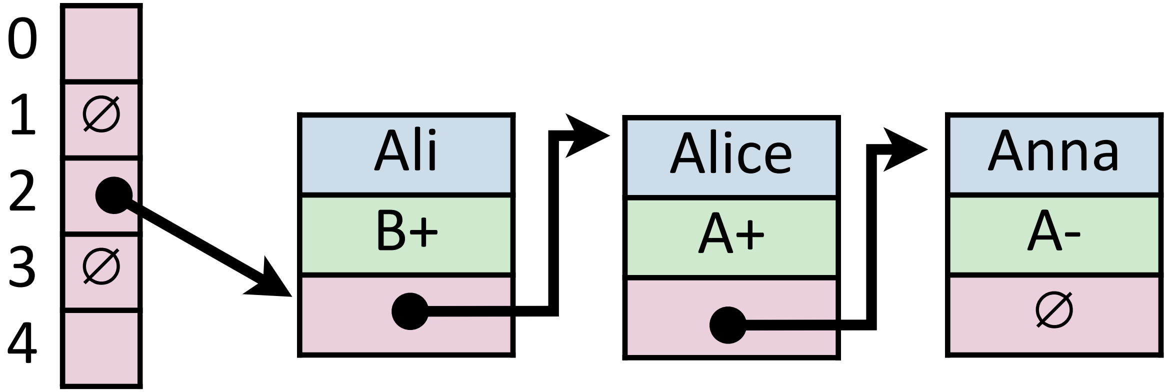
Open vs Closed Hashing

Addressing hash collisions depends on your storage structure.

- **Open Hashing:** store *k,v* pairs externally

Such as a linked list

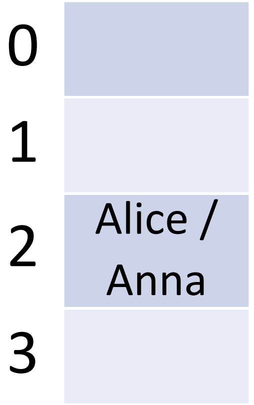
Resolve collisions by adding to list



- **Closed Hashing:** store *k,v* pairs in the hash table

Everything stored in one list

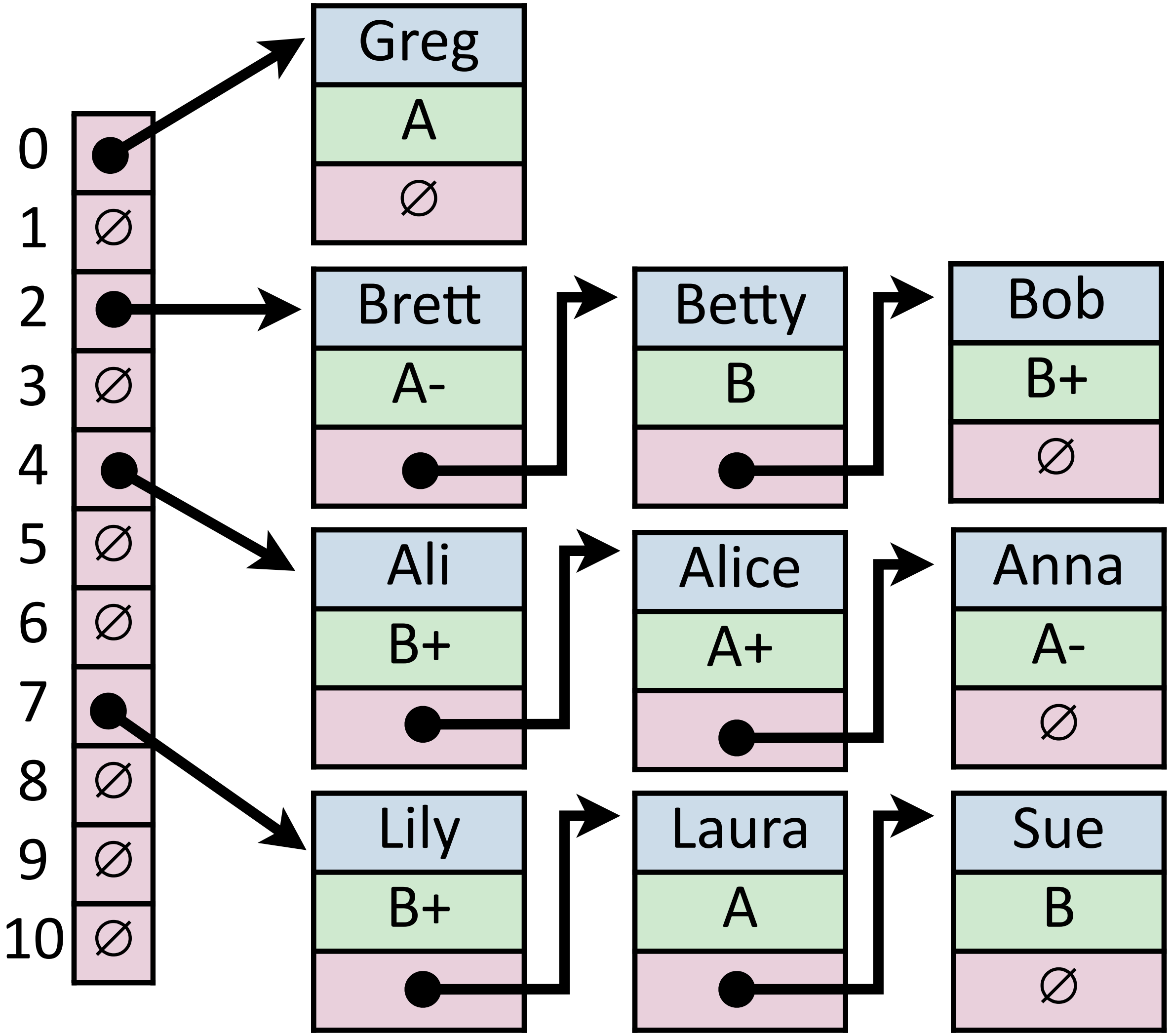
How to store collisions? Unclear!



Hash Table (Separate Chaining)

Linked List InsertFront() — O(1)

Key	Value	Hash
Bob	B+	2
Anna	A-	4
Alice	A+	4
Betty	B	2
Brett	A-	2
Greg	A	0
Sue	B	7
Ali	B+	4
Laura	A	7
Lily	B+	7



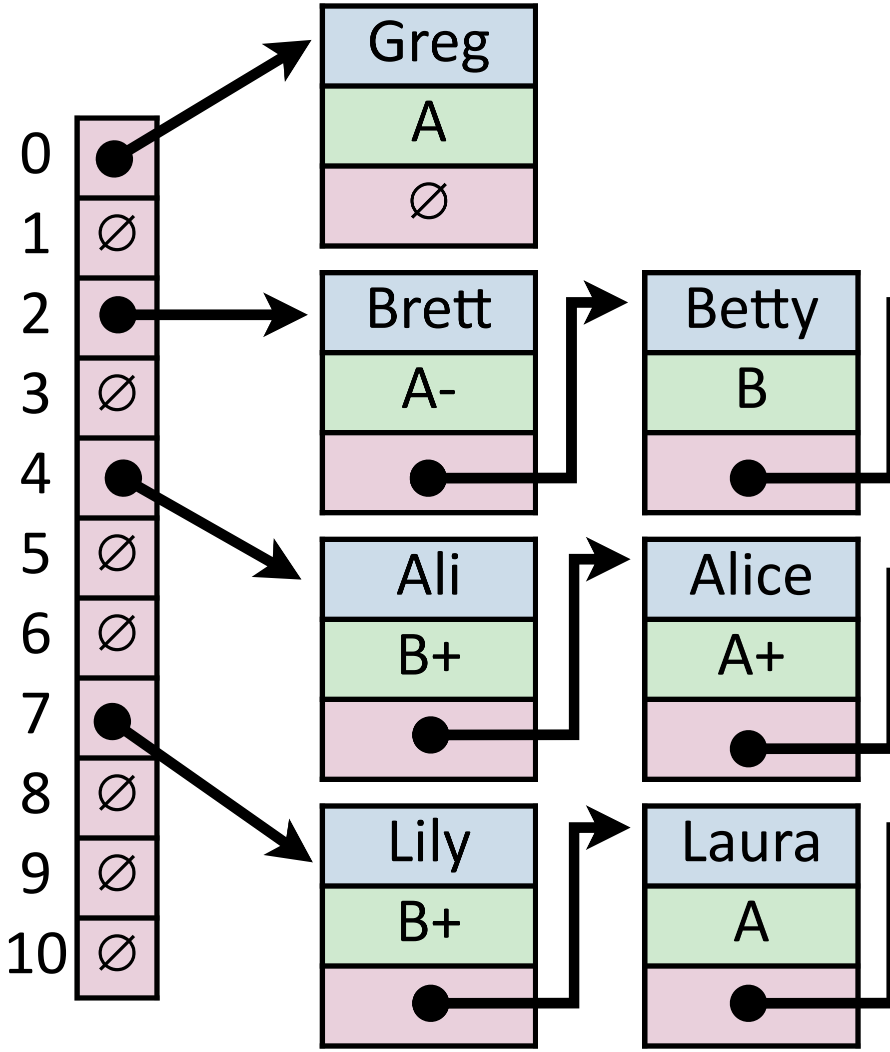
Hash Table (Separate Chaining)

For hash table of size *m* and *n* elements:

Find runs in: O(n), worst case everything collides

Insert runs in: O(1), if hash function is O(1)

Remove runs in: O(n), worst case everything one list



Hash Table

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

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

Simple Uniform Hashing Assumption SUHA is an assumption

(Assume our dataset hashes optimally)

2) Create a ***universal hash function family:*** This is real world SUHA

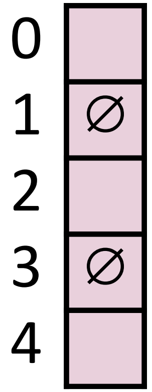
Given a collection of hash functions, pick one randomly

Like **random quicksort** if pick of hash is random, good expectation!

Simple Uniform Hashing Assumption

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

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



Uniform: All keys are equally likely to hash anywhere

$$Pr(h[k_1] = 0) = \frac{1}{m}$$

Independent: All keys hash independently of each other

Simple Uniform Hashing Assumption

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

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

Uniform: All keys equally likely to hash to any position

$$Pr(h[k_1]) = \frac{1}{m}$$

Independent: All key's hash values are independent of other keys

$$Pr(h[k_1] = i \mid h[k_2] = i) = Pr(h[k_1] = i) = \frac{1}{m}$$

Separate Chaining Under SUHA

Table Size: m

Num objects: n

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

$\alpha_j =$ expected # of items hashing to position j

$$\alpha_j = \sum_i H_{i,j}$$

$$H_{i,j} = \begin{cases} 1 \text{ if item i hashes to j} \\ 0 \text{ otherwise} \end{cases}$$

Direct Proof! Count expected # of items

Separate Chaining Under SUHA

Table Size: m

Num objects: n

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

$\alpha_j =$ expected # of items hashing to position j

$$\alpha_j = \sum_i H_{i,j}$$

$$H_{i,j} = \begin{cases} 1 \text{ if item i hashes to j} \\ 0 \text{ otherwise} \end{cases}$$

$$E[\alpha_j] = E\Big[\sum_i H_{i,j}\Big]$$

Separate Chaining Under SUHA

Table Size: m

Num objects: n

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

$\alpha_j =$ expected # of items hashing to position j

$$\alpha_j = \sum_i H_{i,j}$$

$$H_{i,j} = \begin{cases} 1 \text{ if item i hashes to j} \\ 0 \text{ otherwise} \end{cases}$$

$$E[\alpha_j] = E[\sum_i H_{i,j}] = [\sum_i E(H_{i,j})]$$

$$E[\alpha_j] = \sum_i Pr(H_{i,j} = 1) * 1 + Pr(H_{i,j} = 0) * 0$$

Sum of probability * value

Separate Chaining Under SUHA

Table Size: m

Num objects: n

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

$\alpha_j =$ expected # of items hashing to position j

$$\alpha_j = \sum_i H_{i,j} \qquad H_{i,j} = \begin{cases} 1 \text{ if item i hashes to j} \\ 0 \text{ otherwise} \end{cases}$$

$$E[\alpha_j] = E[\sum_i H_{i,j}]$$

$$E[\alpha_j] = \sum_i Pr(H_{i,j} = 1) * 1 + Pr(H_{i,j} = 0) * 0$$

$$E[\alpha_j] = n * Pr(H_{i,j} = 1)$$

Because we have n objects, sum is n * single object

Separate Chaining Under SUHA

Table Size: m

Num objects: n

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

α_j = expected # of items hashing to position j

$$\alpha_j = \sum_i H_{i,j}$$

$$H_{i,j} = \begin{cases} 1 \text{ if item i hashes to j} \\ 0 \text{ otherwise} \end{cases}$$

$$E[\alpha_j] = E[\sum_i H_{i,j}]$$

$$Pr[H_{i,j} = 1] = \frac{1}{m}$$

Under SUHA, above probability is true!

$$E[\alpha_j] = n * Pr(H_{i,j} = 1)$$

Separate Chaining Under SUHA

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

Table Size: m

Num objects: n

α_j = expected # of items hashing to position j

$$\alpha_j = \sum_i H_{i,j}$$

$$H_{i,j} = \begin{cases} \text{1 if item i hashes to j} \\ \text{0 otherwise} \end{cases}$$

$$E[\alpha_j] = E[\sum_i H_{i,j}]$$

$$Pr[H_{i,j} = 1] = \frac{1}{m}$$

$$E[\alpha_j] = n * Pr(H_{i,j} = 1)$$

$$\mathbf{E}[\alpha_\mathbf{j}] = \frac{\mathbf{n}}{\mathbf{m}}$$

Separate Chaining Under SUHA

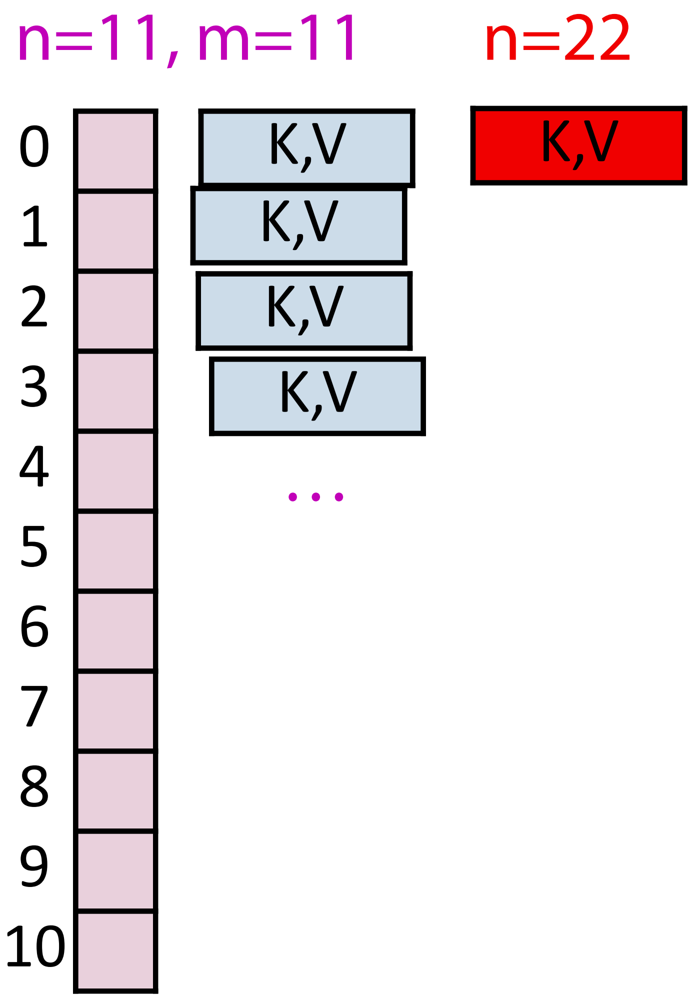
Under SUHA, a hash table of size *m* and *n* elements: **Let α = n/m**

Find runs in: $O(1+\alpha)$.

Insert runs in: $O(1)$.

Remove runs in: $O(1+\alpha)$.

We control what α is!



Collision Handling: Probe-based Hashing

S = { 1, 8 , 15}

|S| = n

h(k) = k % 7

|Array| = m

8%7 = 1

Want to put 8 in "**the next available space**"

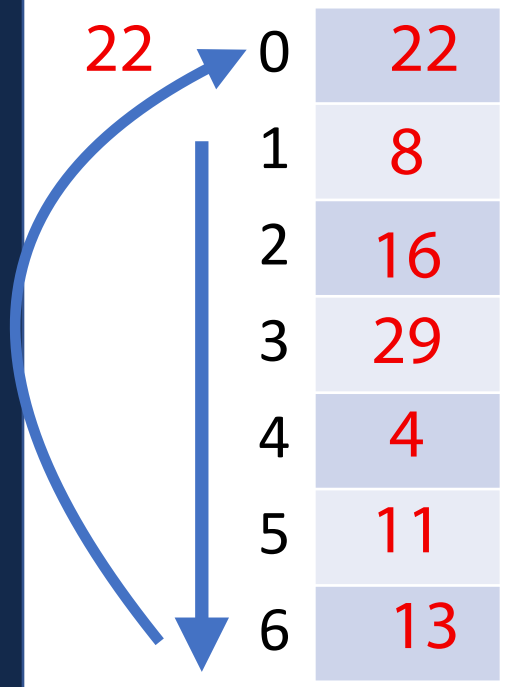
What is a good 'next available'?

(Example of closed hashing)

Collision Handling: Linear Probing

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

h(k) = k % 7 **|Array| = m**



29%7 = 1
29%7+1 = 2
29%7+2 = 3

11%7=4
11%7+1=5

h(k, i) = (k + i) % 7

Try h(k) = (k + 0) % 7, if full…

Try h(k) = (k + 1) % 7, if full…

Try h(k) = (k + 2) % 7, if full…

Try …

22%7 = 1
22%7+1, +2, +3, +4, +5, +6

(Example of closed hashing)

Collision Handling: Linear Probing

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

|S| = n

h(k, i) = (k + i) % 7

|Array| = m

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

`_find(29)`

1) Hash the input key [h(29)=1]

2) Look at hash value (address) position

If present, great! Done!

If not there…

Look at **next available space**

Stop when:

1) We find the object we are looking for

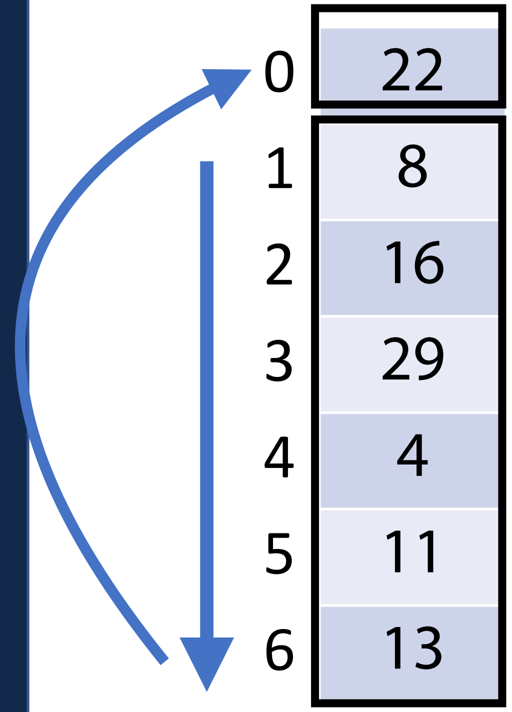
2)

(Example of closed hashing)

Collision Handling: Linear Probing

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

h(k, i) = (k + i) % 7 **|Array| = m**



`_find(30)`

1) Hash the input key [h(29)=1]

2) Look at hash value (address) position

If present, great! Done!

If not there…

Look at **next available space**

Stop when:

1) We find the object we are looking for
2) We have searched every position in the array
3)

(Example of closed hashing)

Collision Handling: Linear Probing

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

h(k, i) = (k + i) % 7 **|Array| = m**

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

If 30 existed, would have been at 4

`_find(30)`

1) Hash the input key [h(29)=1]

2) Look at hash value (address) position

If present, great! Done!

If not there…

Look at **next available space**

Stop when:

1) We find the object we are looking for
2) We have searched every position in the array
3) We find a blank space

(Example of closed hashing)

Collision Handling: Linear Probing

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

h(k, i) = (k + i) % 7 **|Array| = m**

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

1 bit flag (something was inserted here before)

`_remove(16)`

1) Hash the input key [h(16)=2]
2) Find the actual location (if it exists)
3) Remove the (k,v) at hash value (address)

Don't resize the array! Tombstone!

`_find(29)`

With tombstoning, we can go past 2

Still blank but we know at some point it wasn't

A Problem w/ Linear Probing

Primary Clustering: "Rich get richer"

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

Description: Long clusters of items

What is the probability that the next item (in SUHA) ends up at 6?

It should be 1/m but is it?

If hash to 1, insert at 6
If hash to 2, insert at 6
If hash to 3, insert at 6
…

(Example of closed hashing)

Collision Handling: Quadratic Probing

S = { 16, 8, 4, 13, 29, 12, 22 }

|S| = n

h(k) = k % 7

|Array| = m

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

h(k, i) = (k + i*i) % 7

Try h(k) = (k + 0) % 7, if full...

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

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

Try ...

A Problem w/ Quadratic Probing

Secondary Clustering:

0	**0_1**
1	**0_2**
2	
3	
4	**0_3**
5	
6	
7	
8	
9	**0_4**

Description:

Remedy:

(Example of closed hashing)

Collision Handling: Double Hashing

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

$|S| = n$

$|Array| = m$

$h_1(k) = k \% 7$

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

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

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**

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

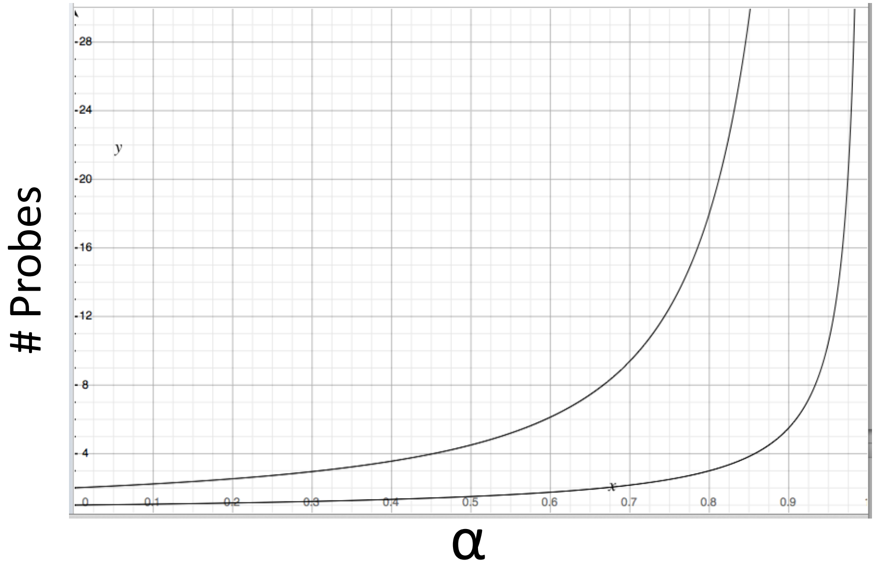
Linear Probing:

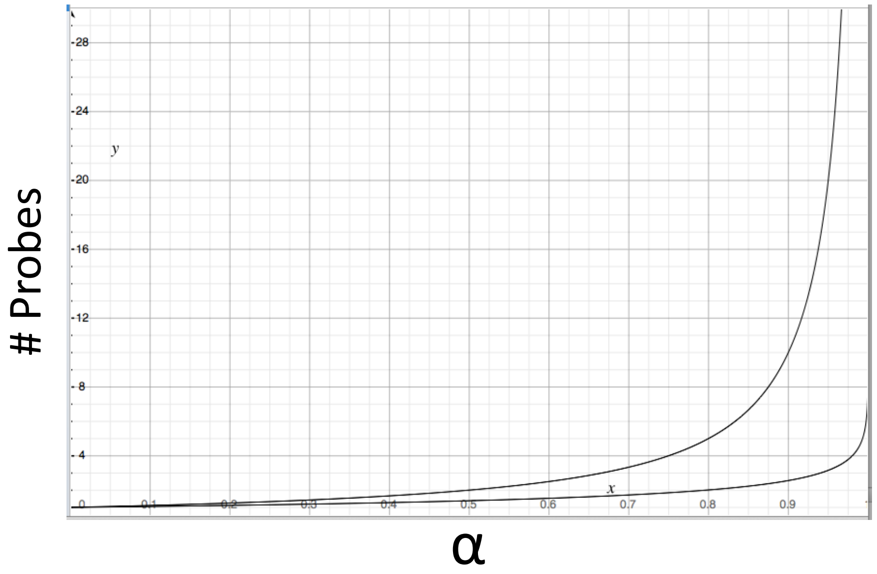
- Successful: **½(1 + 1/(1-α))**
- Unsuccessful: **½(1 + 1/(1-α))²**

Double Hashing:

- Successful: **1/α * ln(1/(1-α))**
- Unsuccessful: **1/(1-α)**

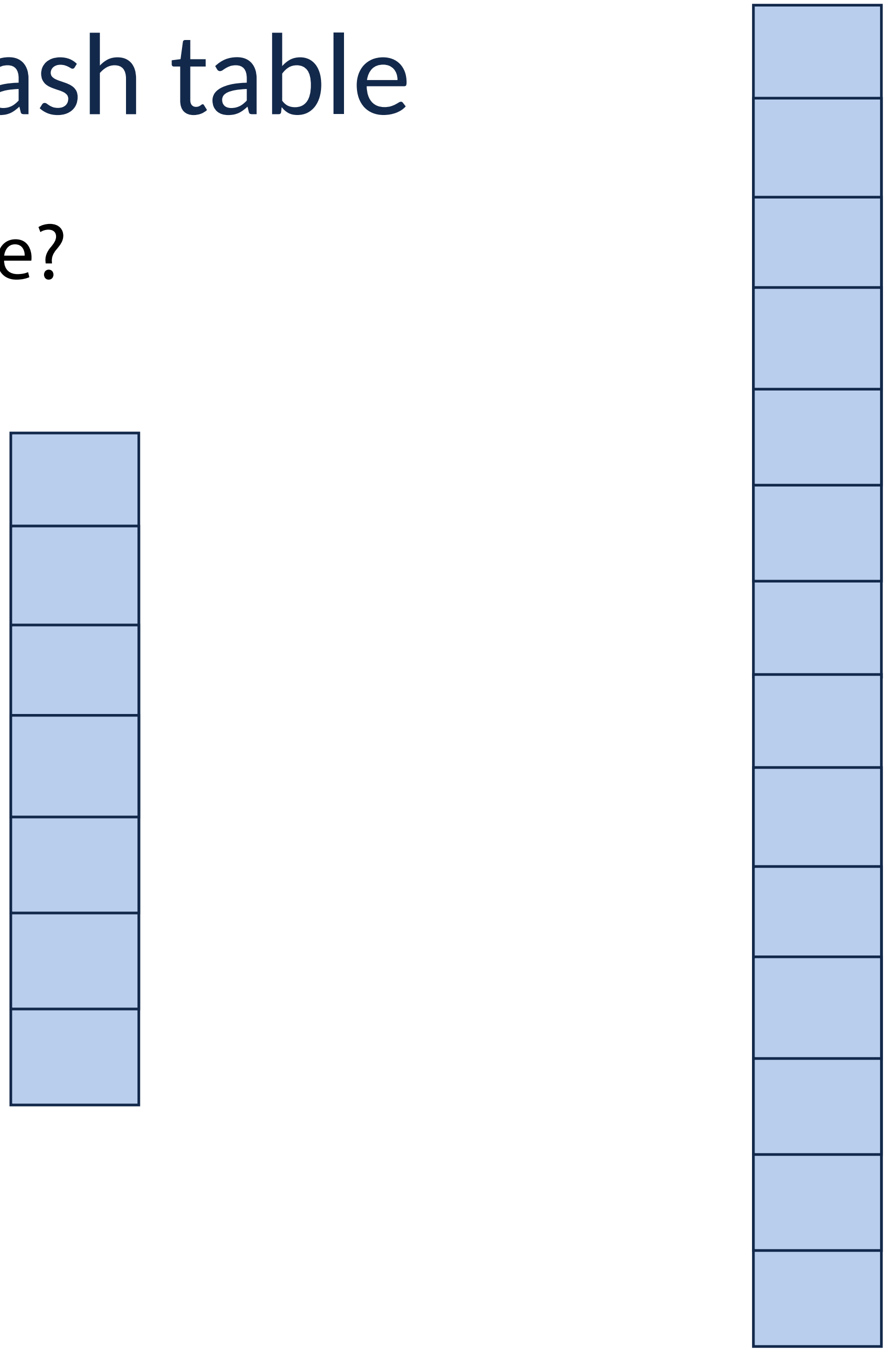
When do we resize?





Resizing a hash table

How do you 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			