Data Structures and Algorithms

Hashing 2

CS 225

Brad Solomon

November 18, 2026



UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Learning Objectives

Review fundamentals of hash tables

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):

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

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

1. A hash function

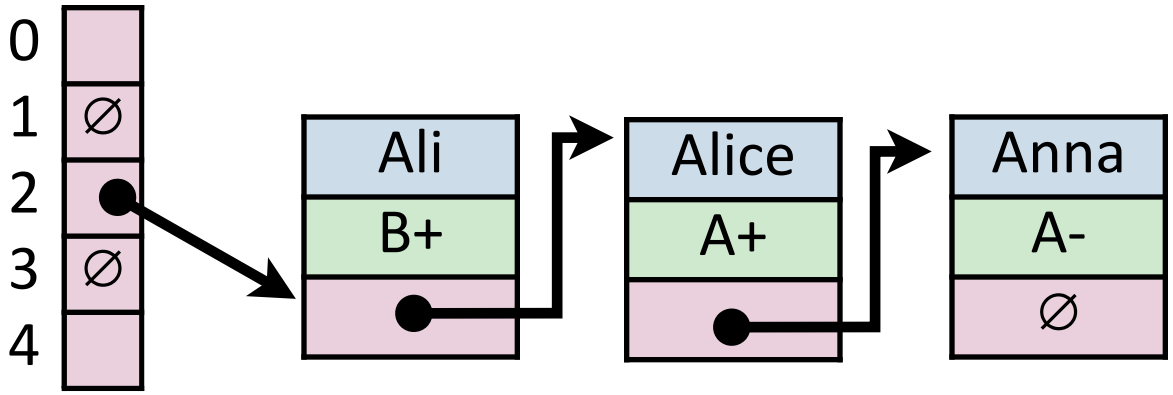
2. A data storage structure

3. A method of addressing *hash collisions*

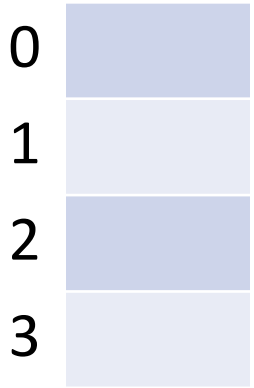
Open vs Closed Hashing

Addressing hash collisions depends on your storage structure.

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

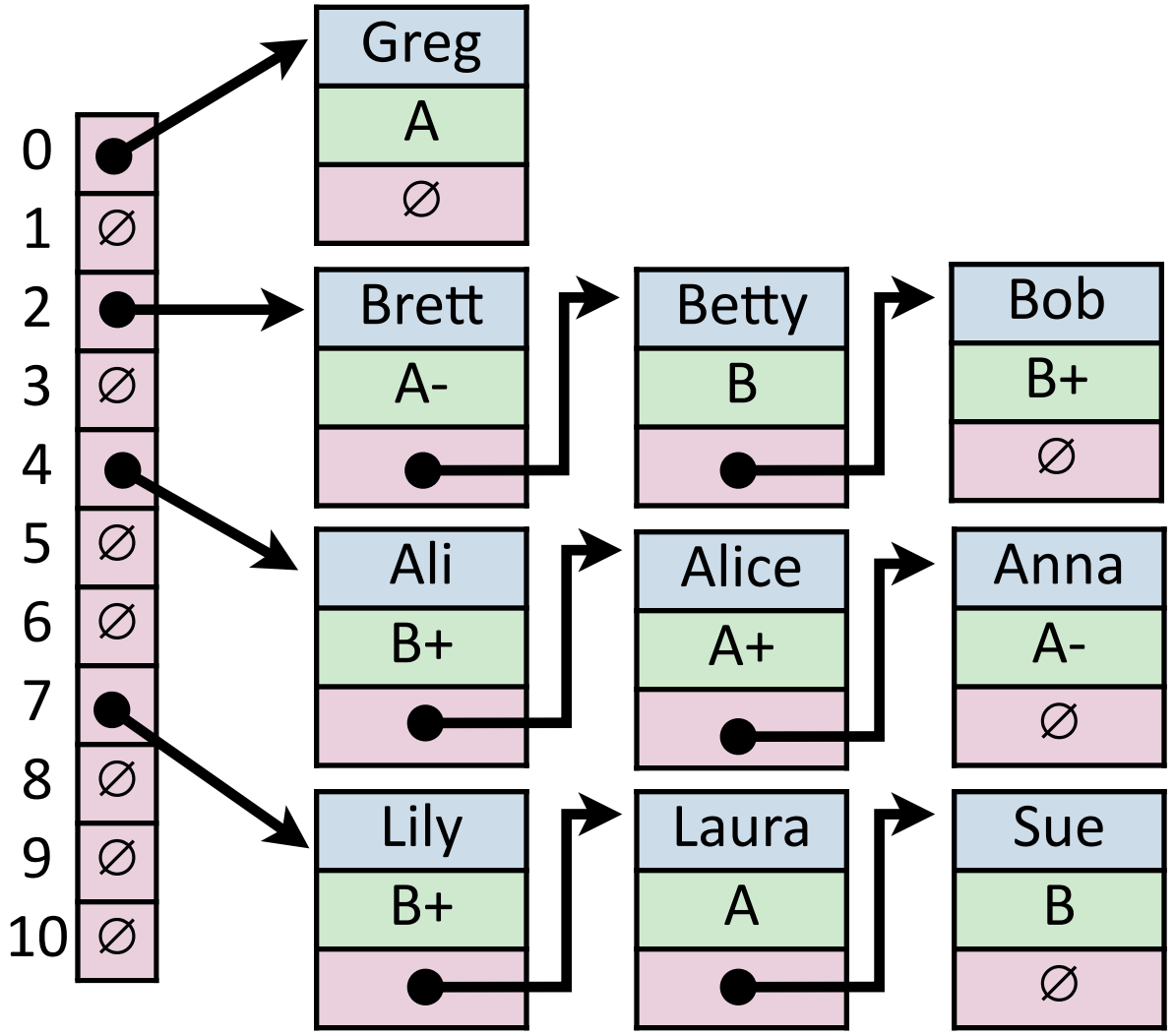


- **Closed Hashing:** store *k,v* pairs in the hash table



Hash Table (Separate Chaining)

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



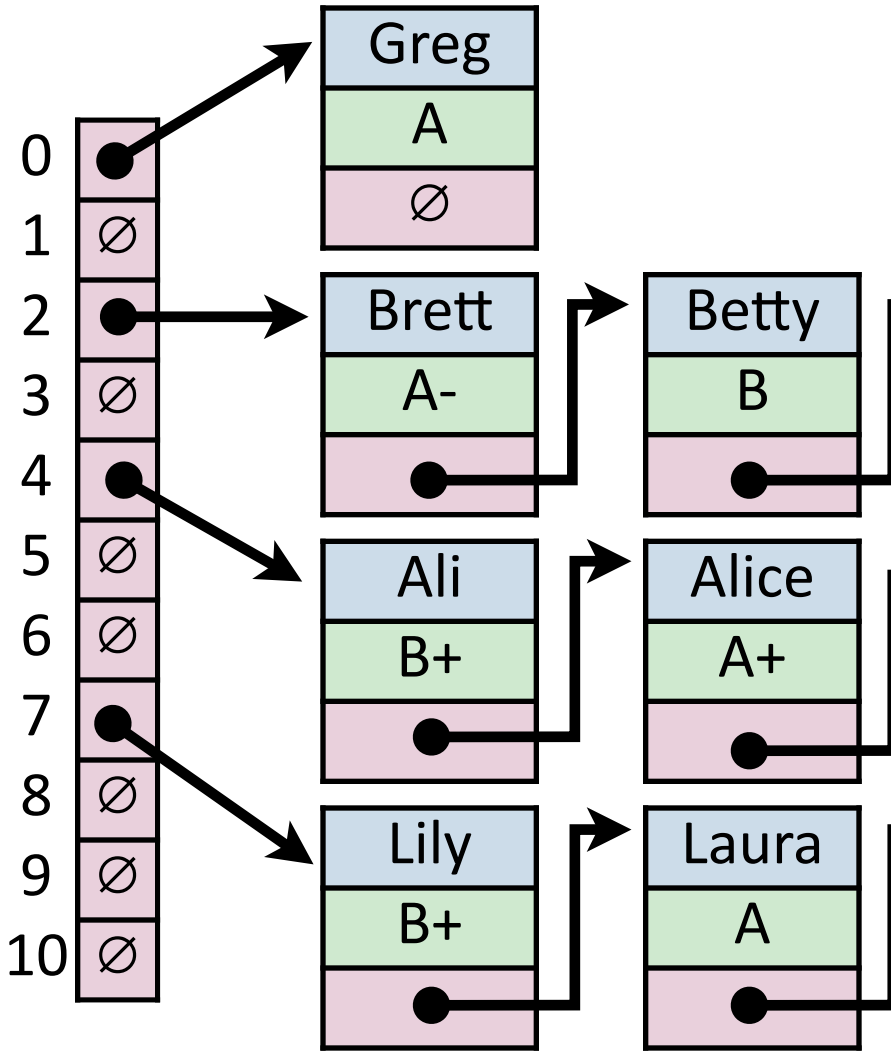
Hash Table (Separate Chaining)

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

Find runs in: _____

Insert runs in: _____

Remove runs in: _____



Hash Table

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

Where is the randomness in a hash table?

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 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

(Assume our dataset hashes optimally)

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

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

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

Uniform:

Independent:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Separate Chaining Under SUHA

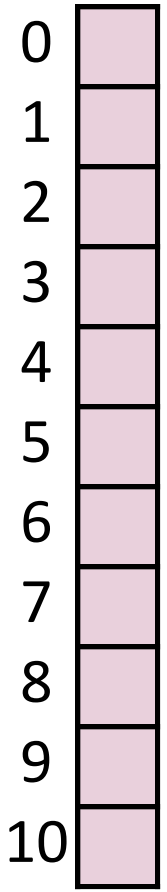


Under SUHA, a hash table of size *m* and *n* elements:

Find runs in: _____.

Insert runs in: _____.

Remove runs in: _____.



(Example of closed hashing)

Collision Handling: Probe-based Hashing

S = { 1, 8 , 15}

|S| = n

h(k) = k % 7

|Array| = m



(Example of closed hashing)

Collision Handling: Linear Probing

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

|S| = n

h(k) = k % 7

|Array| = m



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 …

(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(29)`

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

(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, return (k, v)

If not 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**

`_remove(16)`

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

(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

`_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!

A Problem w/ Linear Probing

Primary Clustering:

0	
1	$\mathbf{1_1}$
2	$\mathbf{1_2}$
3	$\mathbf{3_1}$
4	$\mathbf{1_3}$
5	$\mathbf{3_2}$
6	
7	
8	
9	

Description:

Remedy:

(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	O_1
1	O_2
2	
3	
4	O_3
5	
6	
7	
8	
9	O_4

Description:

Remedy:

(Example of closed hashing)

Collision Handling: Double Hashing

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

$h_1(k) = k \% 7$

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

|S| = n

|Array| = m

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 …

Running Times

(Expectation under SUHA)

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

Open Hashing:

insert: _____.

find/ remove: _____.

Closed Hashing:

insert: _____.

find/ remove: _____.

Running Times (Expectation under SUHA)

Open Hashing: $0 \leq \alpha \leq \infty$

insert: $\frac{1}{\quad}$.

find/ remove: $\frac{1+\alpha}{\quad}$.

Closed Hashing: $0 \leq \alpha < 1$

insert: $\frac{\frac{1}{\quad}}{1-\alpha}$.

find/ remove: $\frac{\frac{1}{\quad}}{1-\alpha}$.

Observe:

- As α increases:

Runtime approaches infinity

- If α is constant:

Runtime is constant

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

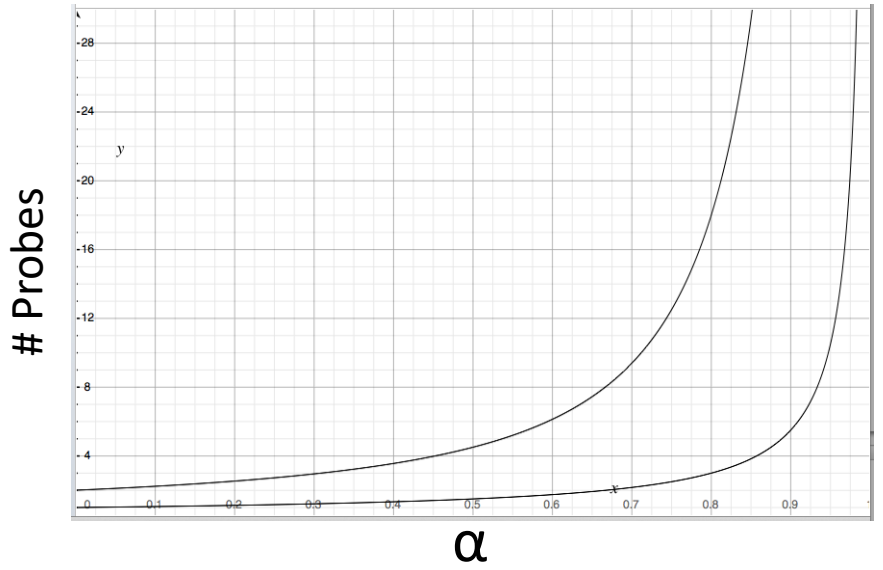
Linear Probing:

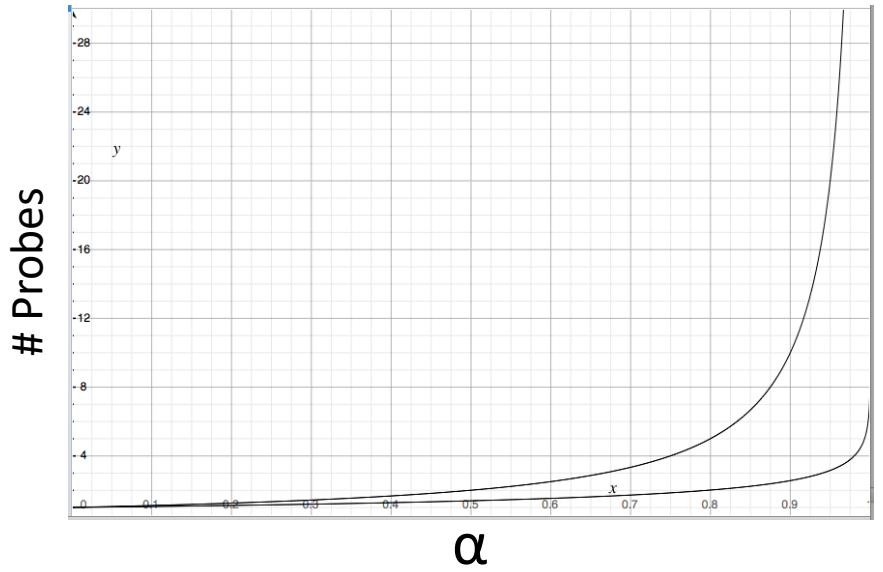
- Successful: **½(1 + 1/(1-α))**
- Unsuccessful: **½(1 + 1/(1-α))²**

Double Hashing:

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

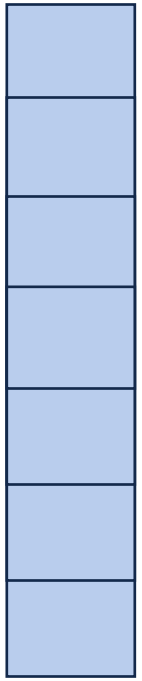
When do we resize?

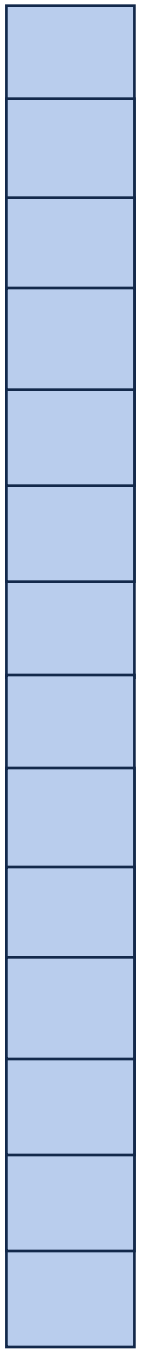




Resizing a hash table

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?

std::map in C++

```
T& map<K, V>::operator[]
pair<iterator, bool> map<K, V>::insert()
iterator map<K, V>::erase()


iterator map<K, V>::lower_bound( const K & );
iterator map<K, V>::upper_bound( const K & );
```

std::unordered_map in C++

```
T& unordered_map<K, V>::operator[]
pair<iterator, bool> unordered_map<K, V>::insert()
iterator unordered_map<K, V>::erase()
```

~~`iterator map<K, V>::lower_bound( const K & );`~~

~~`iterator map<K, V>::upper_bound( const K & );`~~

```
float unordered_map<K, V>::load_factor();
void unordered_map<K, V>::max_load_factor(float m);
```

Running Times

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