

Set ADT Implementations

Note Title

2/19/2008

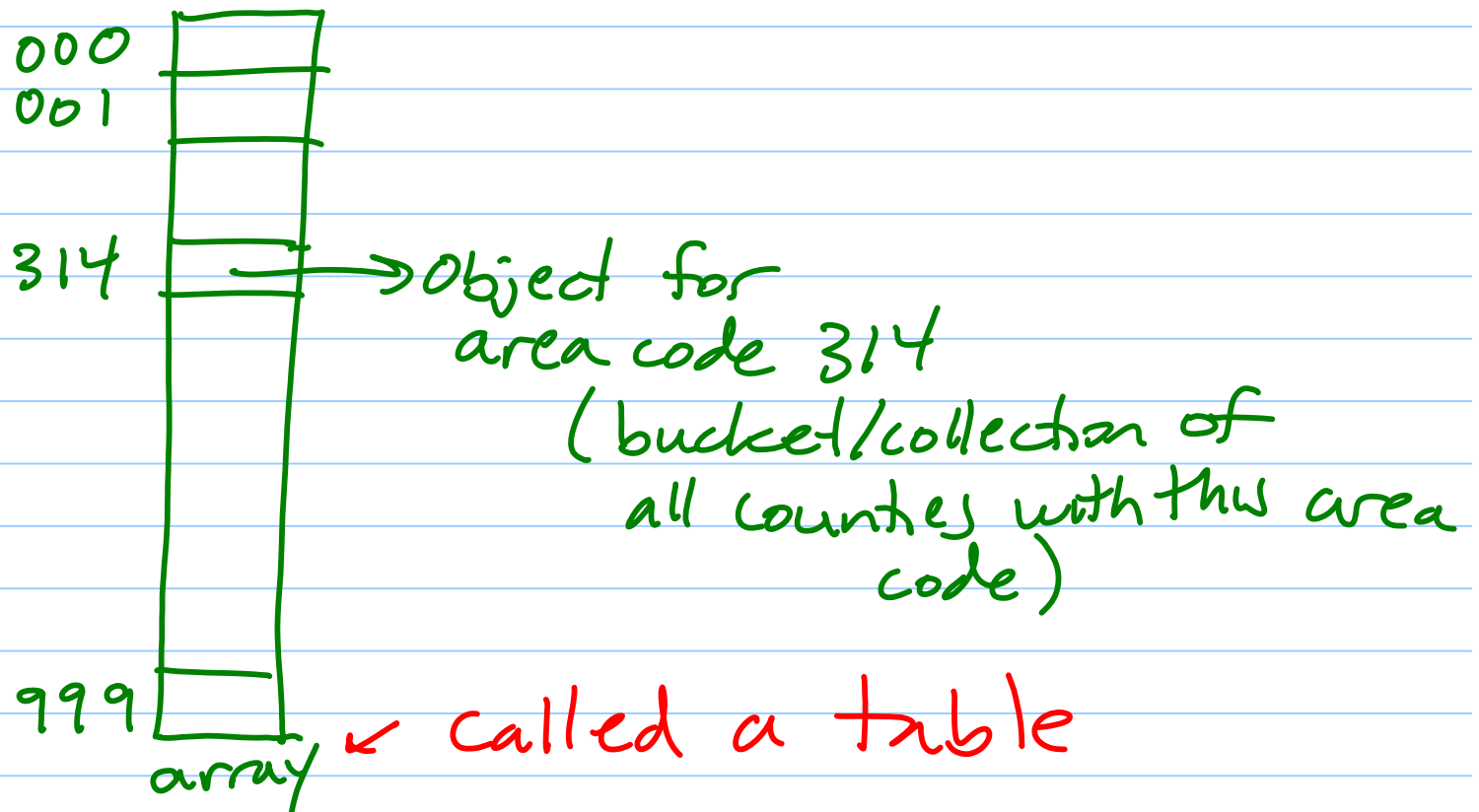
Set holds {elements}

Mapping {tag \rightarrow element}
(Tagged Set, Map)

BucketMapping {tag \rightarrow {elements}}

Direct Addressing

insert, locate, remove



Key here

every element you might possibly
insert into set has a dedicated

index \rightarrow slot in the table

worst-case

insert - table[slot] = element $\Theta(1)$

locate - access table[slot] $\Theta(1)$

remove - table[slot] = null
EMPTY $\Theta(1)$

What is the big limitation?

Size of table is as big as
 \cup { universe of all possible elements
that might be inserted

Consider a univ of 5000 students
where SS # is the id #.

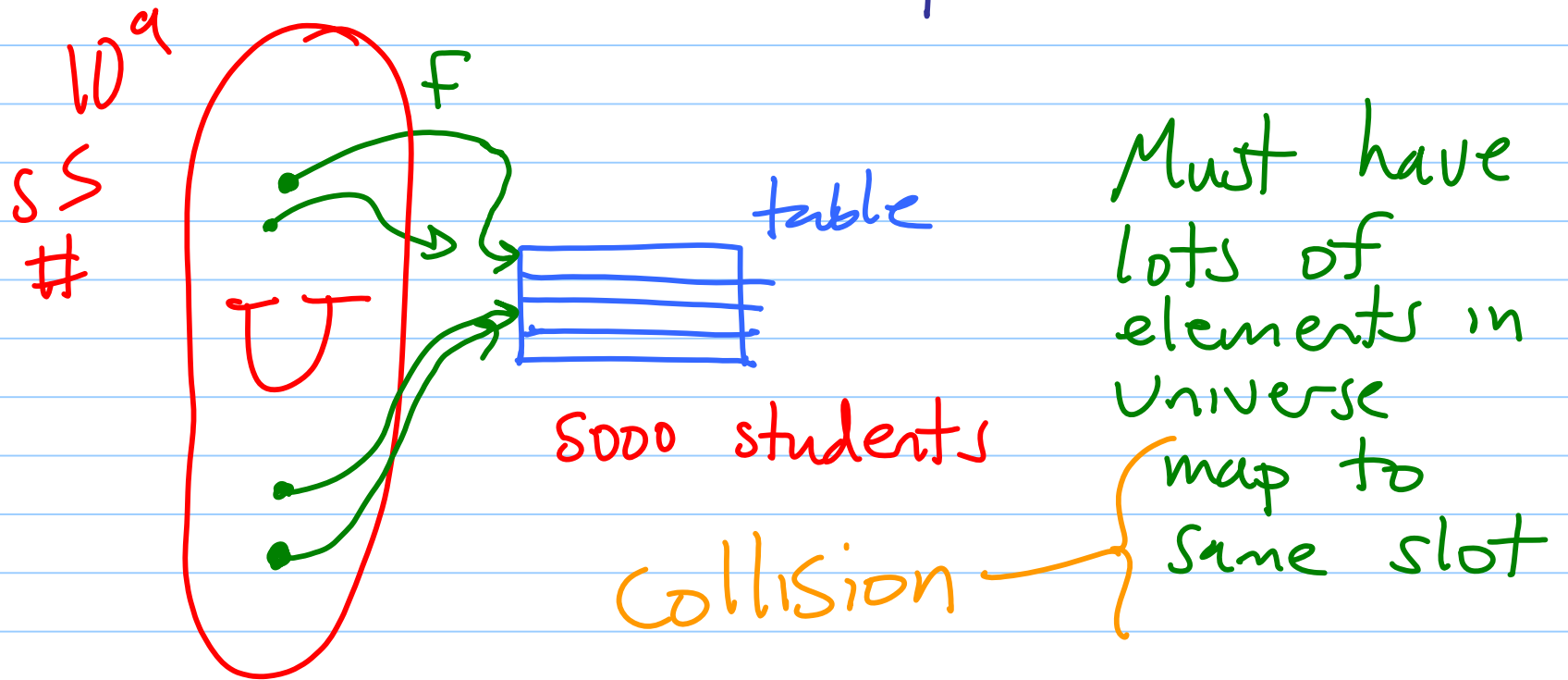
10^9

Direct addressing is only a reasonable choice (in terms of space usage) when

roughly $n > \frac{|U|}{4}$

↑
elements held in set

We want time efficiency of direct addressing but we can't waste so much space



Hash Function

function that maps from hashcode

to $\{0, \dots, m-1\}$

object \rightarrow int

some
integer
from $0, \dots, m-1$

hash table size

Desired property — for each element $x \in U$, $\text{hash}(x, \text{hashcode}())$ is equally likely to be any int in $\{0, \dots, m-1\}$ prob $\frac{1}{m}$

Pick a hash function to be a mathematical function that maps a random integer to $0, \dots, m-1$.

One thought hash code mod m } division method
%

Problem: real data can have patterns in low order digits

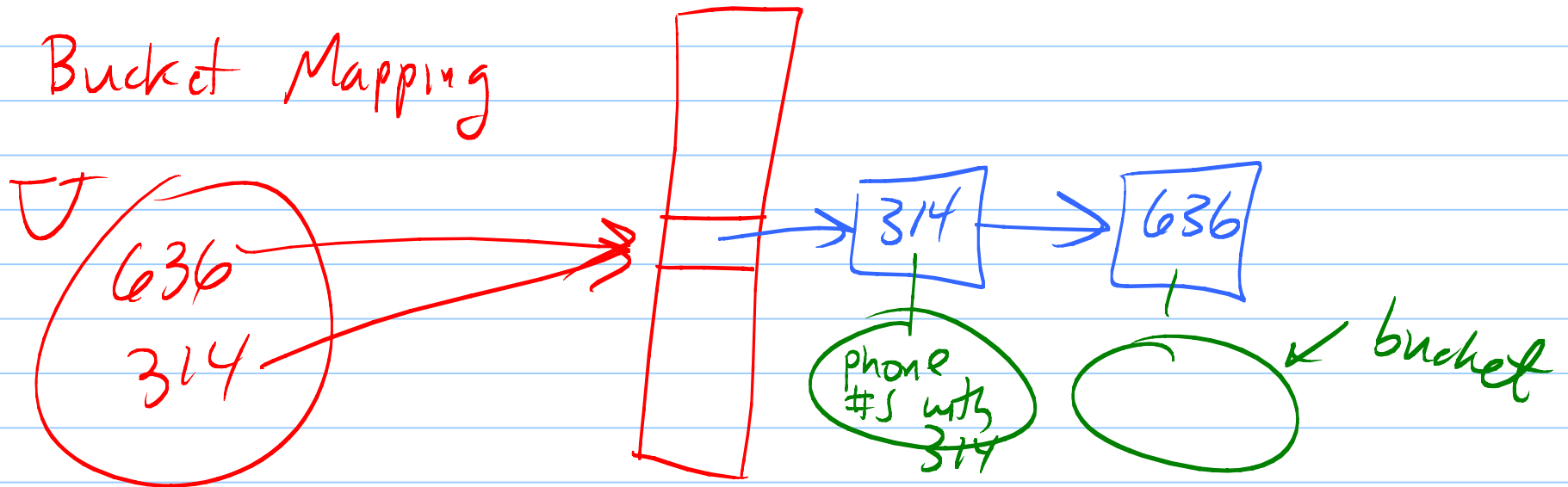
multiplication method - multiply by irrational π

(hash code \times irrational #) rescale
low order
digits
to be
between
 $0 + m - 1$

The remaining question is how to handle collisions.

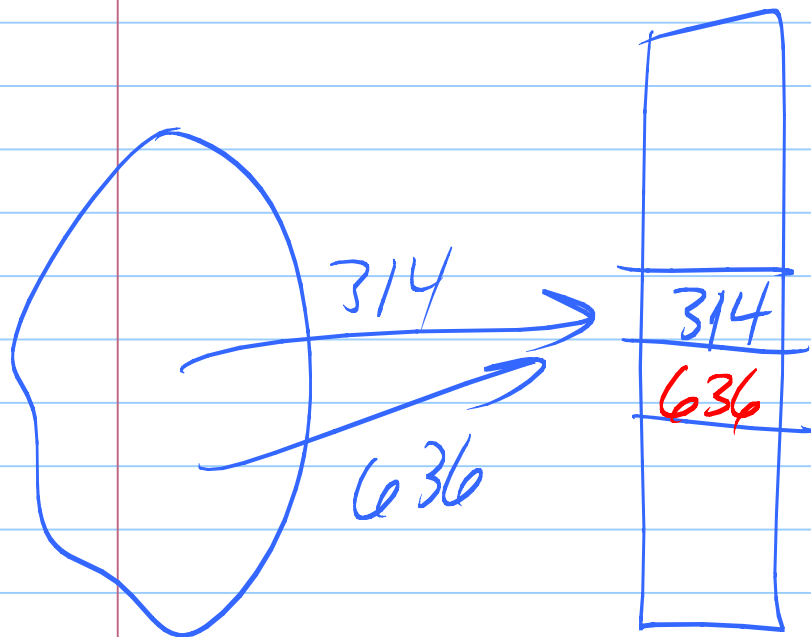
Create a chain ^{list} for each slot

Bucket Mapping



Other option.

Find a different empty slot



cause clustering

Second hash function
to give step size

Open Addressing

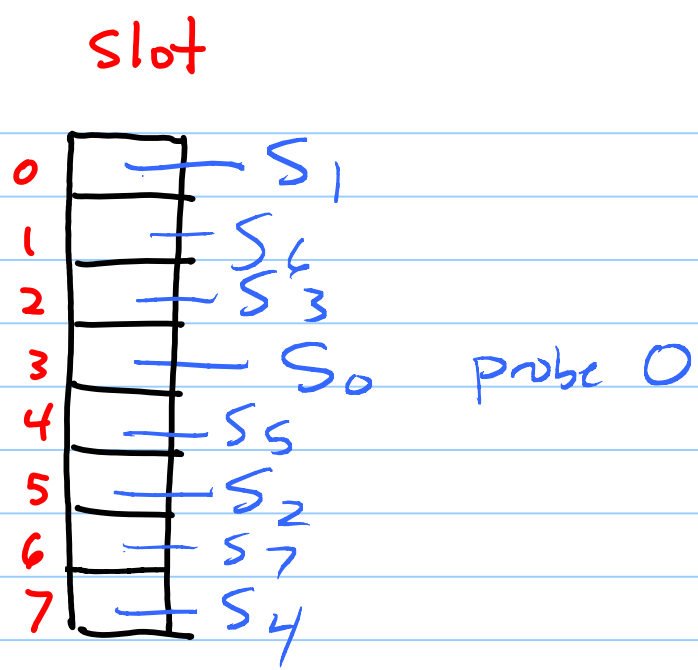
Use first empty slot defined
by probe sequence $\langle S_0, \dots, S_{m-1} \rangle$

Let $x = \text{element.hashCode}()$

$$S_0 = \text{hash}(x)$$

$$i=1, \dots, m-1 \quad S_i = (S_{i-1} + \text{stepHash}(x)) \underset{\substack{\uparrow \\ \text{mod}}}{\%} m$$

} def
of
probe
seq



$m = 8$
 ↖ hash table size

$hash(x) = 3$
 step $hash(x) = 5$

IF $m +$ step size are relatively prime then probe sequence will be a permutation of slots

Provided code makes m
a power of 2

make step size odd

Locating an element - Go through hash table in order of the probe sequence until you either reach the desired element or an empty (unused) slot is reached

element is not in the set

Adding an element

Go through hash table in order given by probe sequence until the element is found or an empty slot s is reached

take appropriate action

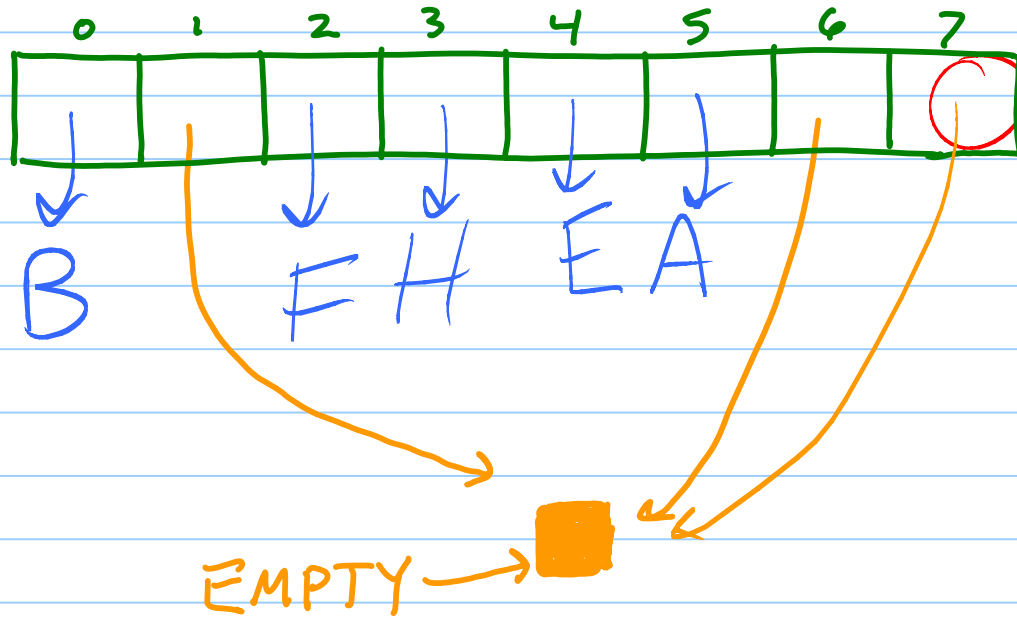
element e was not there & we will place it at slot s $table[s] = e$

Let's look at an example.

element e	A	B	E	F	H	J
$hash(e.hashCode())$	5	0	4	0	5	5
$stepHash(e.hashCode())$	7	1	5	5	3	5

Insert
in order
A, B, E, F,
H, J

table
 $m=8$



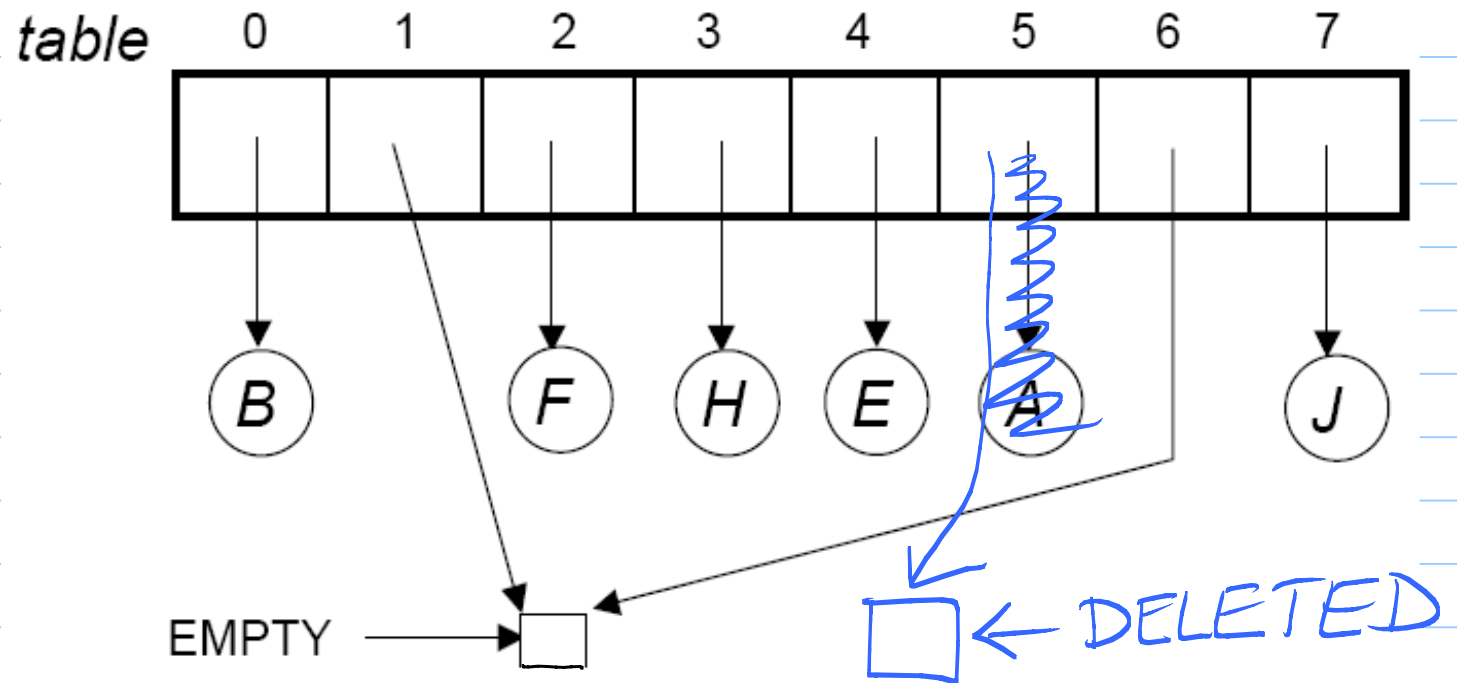
Care must be selected in how m
and $\text{stepHash}(x)$ relate. Why?

How can you remove an element?

Let's delete A.

Now search for F.

element e	A	B	E	F	H	J
$hash(e.hashCode())$	5	0	4	0	5	5
$stepHash(e.hashCode())$	7	1	5	5	3	5



Deleting an element

Problem: hash table can fill up with slots
"marked as deleted" (ref to DELETED)

Partly address this problem by re-using a
deleted slot along probe sequence when
inserting a new element.

In a internal locate method (to check if
element is in set), remember the 1st
DELETED slot found + return that (or
first empty slot if there were no deleted slots)

*only stop if element found
or reach EMPTY slot*

$\alpha = \text{load factor} = \frac{\text{Fraction of slots in use}}{\text{Total slots}}$

n # tags/element
mapping set
bucket collection

$$\frac{n+d}{m}$$

m hash table size

d # slots "marked as deleted"

Actual load versus target load

$$\text{Load factor } \alpha = \frac{n+d}{m}$$

elements in Set
slots marked as deleted
table size

During an unsuccessful search, α is fraction of slots that will cause search to continue.

desired load factor α_d (e.g., $\frac{1}{2}$)

actual load is current value of $\frac{n+d}{m}$

Goal: Keep α ^{actual load} close to α_* ^{desired load}

Limit frequency of resizing ← expensive

Double table size (m) when

α reaches $\frac{1 + \alpha_*}{2}$ } halfway between $\alpha_* + 1$

when expected search cost is twice as large as desired

$\alpha_* = 1/2$, resize when $\alpha = 3/4$

hash functions change + you must rebuild by re-inserting all elements (in order) go through slots, move to next if EMPTY or DELETED, reinsert elements in new table

The hash table could be oversized
(+ cluttered with deleted slots)

half table size when $\frac{n}{m}$ drops
down to $\frac{2}{2}$

reduce hash table size by a
factor of 2.

Analysis

$$E[\text{# probes in an unsuccessful search}] = 1 \cdot \frac{\text{prob. probe 0 occurs}}{\text{occurs}} + 1 \cdot \frac{\text{prob. probe 1 occurs}}{\text{occurs}} + \dots$$

$$\leq 1 + \alpha + \alpha^2 + \alpha^3 + \dots$$

$$\sum_{i=0}^{\infty} \alpha^i =$$

$$\frac{1}{1-\alpha}$$

$\alpha = \frac{1}{2}$,	<u>2</u>
$\alpha = \frac{1}{4}$,	$\frac{4}{3}$
$\alpha = \frac{3}{4}$,	4

expect #
probes

$$E \left[\begin{array}{l} \# \text{ probes in a} \\ \text{successful search} \end{array} \right] = \frac{1}{\alpha} \ln \frac{1}{1-\alpha}$$

Lab 2

acaacg ccg ttacg acc) sequences
gaca --- of k
in common

bucket mapping

string
k-mer \rightarrow { offsets }

