CS61B Lecture #22: Hashing

Speeding Up Searching

h is OK for small data sets, bad for large.

rch would be OK *if* we could rapidly narrow the search ns.

t in constant time we could put any item in our data set cred *bucket,* where # buckets stays within a constant keys.

that buckets contain roughly equal numbers of keys.

would be constant time in number of comparisons.

Hash functions

nust have way to convert key to bucket number: a *hash*

g/ 2a a mixture; a jumble. *b* a mess." *Concise Oxford y, eighth edition*

ata items.

Longs, evenly spread over the range $0..2^{63}-1$.

keep maximum search to $L = 2$ items.

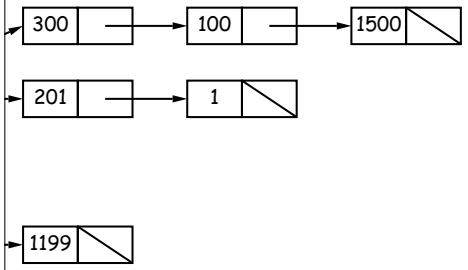
function $h(K) = K\%M$, where $M = N/L = 100$ is the buckets: $0 \le h(K) < M$.

2, 433, and 10002332482 go into different buckets, 0210, and 210 all go into the same bucket.

External chaining

buckets.

is a list of data items.



ts have same length, but average is $N/M = L$, the *load*

hash function must avoid too many *collisions:* keys that ual values.

ching the Chains: Open Addressing

e data item in each bucket.

is a collision, and bucket is full, just use another.

to do this:

bes: If there is a collision at $h(K)$, try $h(K)+m$, $h(K)+$ wrap around at end), until you find an empty bucket.

probes: $h(K)+1\cdot m$, $h(K)+2^2\cdot m$, $h(K)+3^2\cdot m$, ...

shing: $h(K)+h'(K)$, $h(K)+2h'(K)$, etc.

$K) = K\%M$, with $M = 10$, linear probes with $m = 1$.

11, 3, 102, 9, 18, 108, 309 to empty table.

2	11	3	102	309		18	9

et slow, even when table is far from full.

ature on this technique, but

just settle for external chaining.

Filling the Table

y to be) constant-time lookup, need to keep #buckets nt factor of #items.

ble when load factor gets higher than some limit.

ust *re-hash* all table items.

eration constant time per item,

ng table size each time, get constant *amortized* time and lookup

hat is, that our hash function is good).

Functions: Other Data Structures I

List, LinkedList, etc.) are analagous to strings: e.g.,

```
= 1; Iterator i = list.iterator();
hasNext()) {
 obj = i.next();
ode =
ashCode
bj==null ? 0 : obj.hashCode());
```

e spent computing hash function by not looking at entire mple: look only at first few items (if dealing with a List t).

collisions, but does *not* cause equal things to go to dif- ts.

Hash Functions: Strings

"$s_0 s_1 \cdots s_{n-1}$" want function that takes all characters sitions into account.

g with $s_0 + s_1 + \ldots + s_{n-1}$?

Java uses

$$h(s) = s_0 \cdot 31^{n-1} + s_1 \cdot 31^{n-2} + \ldots + s_{n-1}$$

odulo 2^{32} as in Java int arithmetic.

o a table index in $0..N-1$, compute h(s)%N (but *don't* e that is multiple of 31!)

to compute as you might think; don't even need multipli-

```
= 0;
i = 0; i < s.length (); i += 1)
r << 5) - r + s.charAt (i);
```


Identity Hash Functions

ress of object ("hash on identity") if distinct (!=) ob- ver considered equal.

Won't work for Strings, because .equal Strings could nt buckets:

```
= "Hello",
= H + ", world!",
= "Hello, world!";
```

als(S2), but S1 != S2.

Functions: Other Data Structures II

defined data structures ⇒ recursively defined hash

, on a binary tree, one can use something like

```
 == null)
eturn 0;
return someHashFunction (T.label ())
       ^ hash(T.left ()) ^ hash(T.right ());
```


ial Case: Monotonic Hash Functions

hash function is *monotonic:* either nonincreasing or ing.

y $k_1 > k_2$, then $h(k_1) \geq h(k_2)$.

time-stamped records; key is the time.

unction is to have one bucket for every hour.

, you *can* use a hash table to speed up range queries

applied to strings? When would it work well?

What Java Provides

ct, is function hashCode().

eturns the identity hash function, or something similar. OK as a default?]

it for your particular type.

given on last slide, is overridden for type String, as well s in the Java library, like all kinds of List.

ashtable, HashSet, and HashMap use hashCode to give -up of objects.

```
yType,ValueType> map =
ashMap<>(approximate size, load factor);
, value);          // Map KEY -> VALUE.
(someKey)          // VALUE last mapped to by SOMEKEY.
ntainsKey(someKey) // Is SOMEKEY mapped?
ySet()             // All keys in MAP (a Set)
```

Characteristics

od hash function, add, lookup, deletion take $\Theta(1)$ time,

es where one looks up *equal* keys.

or range queries: "Give me every name between Martin [Why?]

robably not a good idea for small sets that you rapidly iscard [why?]

Perfect Hashing

of keys is *fixed*.

e hash function might then hash every key to a differ-*rfect hashing*.

, there is no search along a chain or in an open-address the element at the hash value is or is not equal to the

, might use first, middle, and last letters of a string -digit base-26 numeral). Would work if those letters all strings in the set.

e the Java method, but tweak the multipliers until all different results.