Algorithms   Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

3.4 Hash Tables

- basic ideas
- separate chaining
- linear probing
- hash functions
- context

http://algs4.cs.princeton.edu

3.4 Hash Tables

- basic ideas
- separate chaining
- linear probing
- hash functions
- context

ST implementations: summary

implementation	worst-case cost (after N inserts)			average-case cost (after N random inserts)			ordered iteration?	key interface
	search	insert	delete	search hit	insert	delete		
sequential search (unordered list)	N	N	N	N/2	N	N/2	no	`equals()`
binary search (ordered array)	lg N	N	N	lg N	N/2	N/2	yes	`compareTo()`
BST	N	N	N	1.38 lg N	1.38 lg N	?	yes	`compareTo()`
red-black BST	2 lg N	2 lg N	2 lg N	1.00 lg N	1.00 lg N	1.00 lg N	yes	`compareTo()`

Q. Can we do better?

A. Yes, but with different access to the data.

Space vs. Time

quotes	authors
The iron-folding doors of the small-room or oven were opened.	Babbage
How to teach your horse to pretend hes a vicious animal and chase after others, even if he is	Horse_ebooks
Does the body rule the mind or does the mind rule the body? I dunno...	Morrissey

Brute force

- Treat quote as a number.
  - 180 character limit. 600 bits per quote.
- Need array of length 2^{600}, or about 10^{180}.

Seems bad, but if Moore's law were an actual law, it'd only take a millennium.

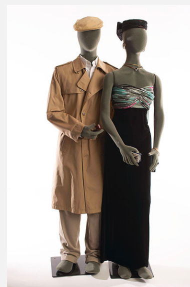
Issues

- Holographic principal provides bound on information density.
  - No more than 1 bit per Planck unit of area.
  - 10^{69} bits per **square** meter of surface area of a sphere.
  - 14 gigaparsec universe contains no more than 10^{122} bits.
- Can also bound information with Bekenstein bound.
- Information density maximized with a black hole.
  - Try to cram more bits than bound: Collapses into black hole.

Spies

Goal: Determining overlap

- Two spies have obtained a large cache of secret documents.
- They want to know which single document they have in common.
  - Must match EXACTLY!
- Can only communicate via slips of paper discretely placed around town.
  - High latency.
  - Low bandwidth.
  - Entire document transmission possible, but very tricky.
- Can coordinate plan before their mission.



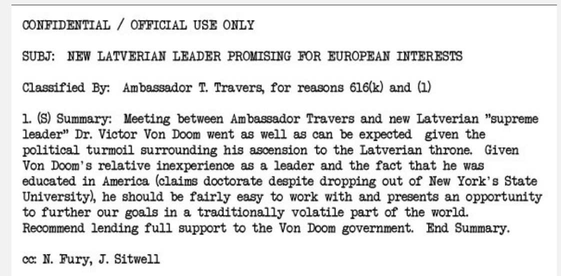
Technique one

- One spy transmits entire text of document to the other.
  - Very slow.

Technique two: Header transmission

Transmit all header

- Each spy transmits only the first 10 characters of each document.
- Issue 1:
  - Not enough to establish equality.
  - Fix:
- New issue:
  - Worst case:



CONFIDENTIAL / OFFICIAL USE ONLY

SUBJ: NEW LATVERIAN LEADER PROMISING FOR EUROPEAN INTERESTS

Classified By: Ambassador T. Travers, for reasons 616(k) and (l)

1. (S) Summary: Meeting between Ambassador Travers and new Latverian "supreme leader" Dr. Victor Von Doom went as well as can be expected given the political turmoil surrounding his ascension to the Latverian throne. Given Von Doom's relative inexperience as a leader and the fact that he was educated in America (claims doctorate despite dropping out of New York's State University), he should be fairly easy to work with and presents an opportunity to further our goals in a traditionally volatile part of the world. Recommend lending full support to the Von Doom government. End Summary.

cc: N. Fury, J. Sitwell

Technique three: Summary transmission

Transmit a summary

-

Hash functions

Essential idea:

- Given a document, **calculate a summary.**
- Transmit summaries.
- If two summaries match, transmit entire document.

Hash functions

- Converts large object into a small one.
- Desired properties:
  - Deterministic.
  - Differ inputs result in different outputs.
  - Easy to compute.

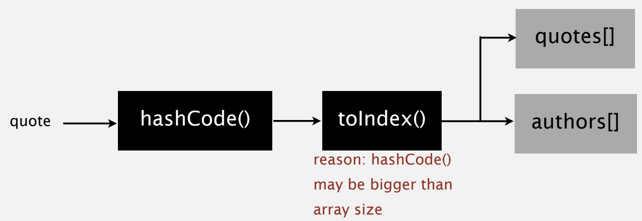
Using hash functions for indexing

Essential idea:

- Given a document, calculate its hash.
- Transmit hashes.
- If two hashes match, transmit entire document.

Storing a quote

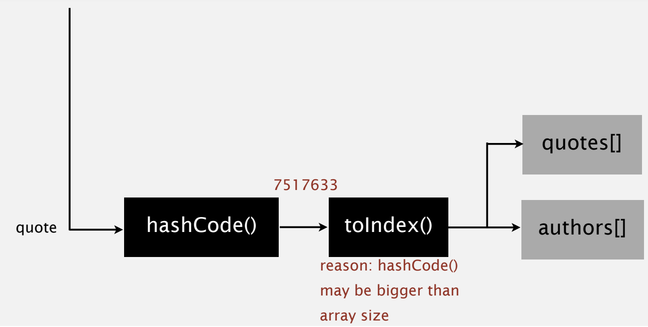
- Maintain quote and author arrays.
- Quote in position i corresponds to author in position i.
- To insert a quote, calculate its hash.
  - Store quote and author at a position determined by its hash.



Example: put

index	quotes[]	authors[]
0	“The iron-folding doors of the small-room or oven were opened.”	Babbage
1		
2		

"By convention there is sweetness, by convention bitterness, by convention color, in reality only atoms and the void." - The Intellect (Democritus)



Modular hashing

Hash code. An `int` between -2^{31} and $2^{31} - 1$.

Hash function. An `int` between `0` and `M - 1` (for use as array index).

typically a prime or power of 2

```
private int hash(Key key)
{  return key.hashCode() % M;  }
```

bug

```
private int hash(Key key)
{  return Math.abs(key.hashCode()) % M;  }
```

1-in-a-billion bug

hashCode() of "polygenelubricants" is -2^{31}

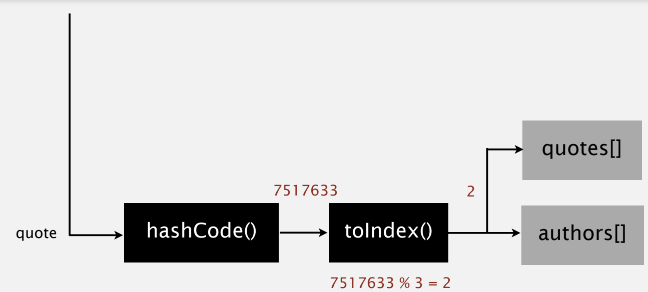
```
private int hash(Key key)
{  return (key.hashCode() & 0x7fffffff) % M;  }
```

correct

Example: put

index	quotes[]	authors[]
0	“The iron-folding doors of the small-room or oven were opened.”	Babbage
1		
2	“By convention there is sweetness, by convention bitterness, by convention color, in reality only atoms and the void.”	Democritus

"By convention there is sweetness, by convention bitterness, by convention color, in reality only atoms and the void." - The Intellect (Democritus)



Symbol table development

First attempt

- See code

Issues

- How do we write a hash function? (later)
- What do we do in the event of a hash collision?
- What do we do when the table becomes full?

http://algs4.cs.princeton.edu

3.4 Hash Tables

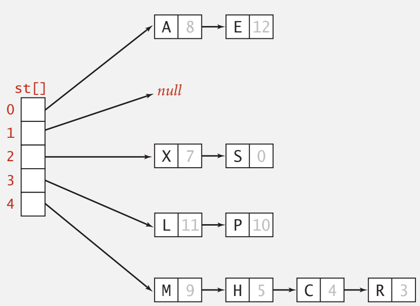
- basic ideas
- separate chaining
- linear probing
- hash functions
- context

Separate chaining symbol table

Use an array of $M < N$ linked lists. [H. P. Luhn, IBM 1953]

- Hash: map key to integer i between 0 and $M - 1$.
- Insert: put at front of i^{th} chain (if not already there).
- Search: need to search only i^{th} chain.

key	hash	value
S	2	0
E	0	1
A	0	2
R	4	3
C	4	4
H	4	5
E	0	6
X	2	7
A	0	8
M	4	9
P	3	10
L	3	11
E	0	12



Put

```
public void put(Key key, Value val) {
    int i = hash(key);
    Node firstNodeInBucket = st[i];
    st[i] = new Node(key, val, firstNodeInBucket);
}
```

```
public void put(Key key, Value val) {
    int i = hash(key);
    Node firstNodeInBucket = st[i];

    for (Node x = firstNodeInBucket; x != null; x = x.next)
        if (key.equals(x.key)) {
            x.val = val;
            return;
        }

    st[i] = new Node(key, val, firstNodeInBucket);
}
```

pollEv.com/jhug text to **37607**

Which put method do you like better?

A. Top [43446]
B. Bottom [43704]

Symbol table development

Sequential Chaining Hash Table

- See code

Issues

- How do we write a hash function? (later)
- What do we do in the event of a hash collision?
- What do we do when the table becomes full?

Performance

- N << M: Constant time get and put.
- N >> M: Linear time.

Resizing

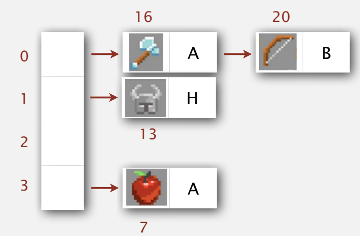
Objective

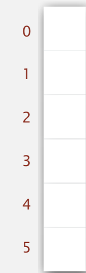
- Keep lists short.
- Don't waste memory on empty lists.

Approach

- Increase size of array when N exceeds some constant factor of M.
- Decrease size of array when N decreases below some constant factor of M.

pollEv.com/jhug   text to **37607**





In which bin will the apple appear after resizing?

0 [9575]
1 [9597]
2 [9609]
3 [9635]
4 [9637]
5 [9643]

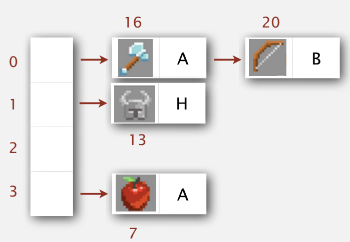
Resizing

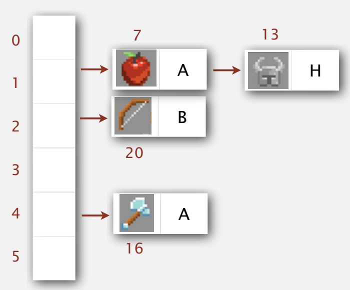
Objective

- Keep lists short.
- Don't waste memory on empty lists.

Approach

- Increase size of array when N exceeds some constant factor of M.
- Decrease size of array when N decreases below some constant factor of M.





Resize

```
private void resize(int size) {
    Node[] newSt = (Node[]) new Object[size];

    for (int i = 0; i < st.length; i++)
        newSt[i] = st[i];

    M = size;
    st = newSt;
}
```

pollEv.com/jhug   text to **37607**

Will the resize method above work correctly?

A. Yes [46372]
B. No [1431]

Symbol table analysis

Sequential Chaining Hash Table

- See code

Performance

- ~~N << M: Constant time get and put.~~
- ~~N >> M: Linear time.~~
- N within a small constant factor of M.

These cases are now impossible.

Analysis

- How full are the bins?
  - Average bin.
  - Worst case bin.

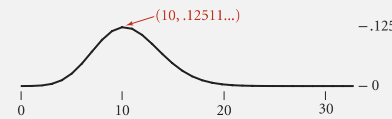
Requires COS 340 math.

Uniform hashing assumption. Each key is equally likely to hash to an integer between 0 and $M-1$.

Analysis of separate chaining

Proposition. Under uniform hashing assumption, prob. that the number of keys in a list is within a constant factor of N/M is extremely close to 1.

Pf sketch. Distribution of list size obeys a binomial distribution.



Binomial distribution $(N = 10^4, M = 10^3, \alpha = 10)$

N/M

equals() and hashCode()

Consequence. Number of probes for search/insert is proportional to N/M.

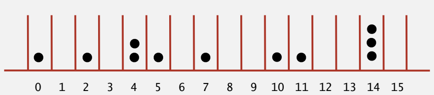
- M too large $\Rightarrow$ too many empty chains.
- M too small $\Rightarrow$ chains too long.
- Typical choice: $M \sim N/5 \Rightarrow$ constant-time ops.

M times faster than sequential search

Other consequences of uniform hashing

Uniform hashing assumption. Each key is equally likely to hash to an integer between 0 and $M-1$.

Bins and balls. Throw balls uniformly at random into M bins.



N proportional to sqrt(M) gives no collisions.

Birthday problem. Expect two balls in the same bin after $\sim \sqrt{\pi M / 2}$ tosses.

Coupon collector. Expect every bin has ≥ 1 ball after $\sim M \ln M$ tosses.

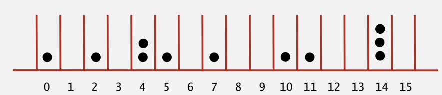
Load balancing. After M tosses, expect most loaded bin has $\Theta(\log M / \log \log M)$ balls.

N proportional to M gives worst case expected performance of log M / log log M.

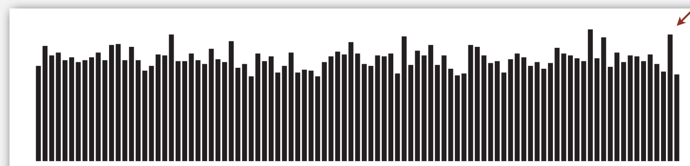
Uniform hashing assumption

Uniform hashing assumption. Each key is equally likely to hash to an integer between 0 and $M-1$.

Bins and balls. Throw balls uniformly at random into M bins.



Expect largest bin to grow as log N / log log N



Hash value frequencies for words in Tale of Two Cities (M = 97)

Java's String data uniformly distribute the keys of Tale of Two Cities

ST implementations: summary

implementation	worst-case cost (after N inserts)			average case (after N random inserts)			ordered iteration?	key interface
	search	insert	delete	search hit	insert	delete		
sequential search (unordered list)	N	N	N	N/2	N	N/2	no	`equals()`
binary search (ordered array)	lg N	N	N	lg N	N/2	N/2	yes	`compareTo()`
BST	N	N	N	1.38 lg N	1.38 lg N	?	yes	`compareTo()`
red-black tree	2 lg N	2 lg N	2 lg N	1.00 lg N	1.00 lg N	1.00 lg N	yes	`compareTo()`
separate chaining	$\Theta(\frac{\log N}{\log \log N})$*	$\Theta(\frac{\log N}{\log \log N})$*	$\Theta(\frac{\log N}{\log \log N})$*	3-5 *	3-5 *	3-5 *	no	`equals()` `hashCode()`

* expected under uniform hashing assumption

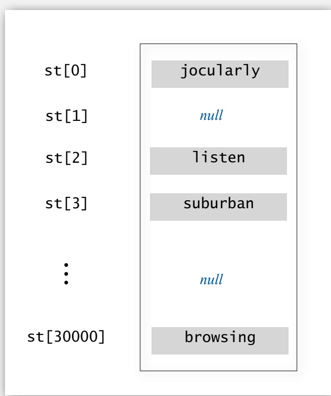
http://algs4.cs.princeton.edu

3.4 Hash Tables

- basic ideas
- separate chaining
- **linear probing**
- hash functions
- context

Collision resolution: open addressing

Open addressing. [Amdahl-Boehme-Rocherster-Samuel, IBM 1953]
When a new key collides, find next empty slot, and put it there.

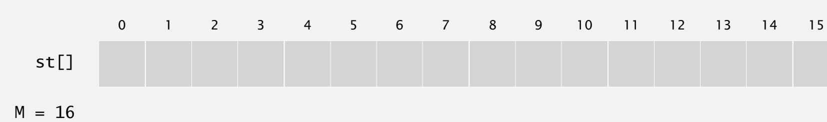


linear probing (M = 30001, N = 15000)

Linear probing hash table demo

Hash. Map key to integer `i` between `0` and `M-1`.
Insert. Put at table index `i` if free; if not try `i+1`, `i+2`, etc.

linear probing hash table



Linear probing hash table demo

Hash. Map key to integer `i` between `0` and `M-1`.

Search. Search table index `i`; if occupied but no match, try `i+1`, `i+2`, etc.

search K

hash(K) = 5

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
st[]	P	M			A	C	S	H	L		E				R	X
M = 16										K						

search miss (return null)

Linear probing hash table summary

Hash. Map key to integer `i` between `0` and `M-1`.

Insert. Put at table index `i` if free; if not try `i+1`, `i+2`, etc.

Search. Search table index `i`; if occupied but no match, try `i+1`, `i+2`, etc.

Note. Array size `M` must be greater than number of key-value pairs `N`.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
st[]	P	M			A	C	S	H	L		E				R	X

M = 16

Linear probing ST implementation

```
public class LinearProbingHashST<Key, Value>
{
   private int M = 30001;
   private Value[] vals = (Value[]) new Object[M];
   private Key[]   keys = (Key[])   new Object[M];

   private int hash(Key key)             {  /* as before */   }

   private void put(Key key, Value val) {  /* next slide */  }

   public Value get(Key key)
   {
      for (int i = hash(key); keys[i] != null; i = (i+1) % M)
         if (key.equals(keys[i]))
             return vals[i];
      return null;
   }

}
```

array doubling and halving code omitted

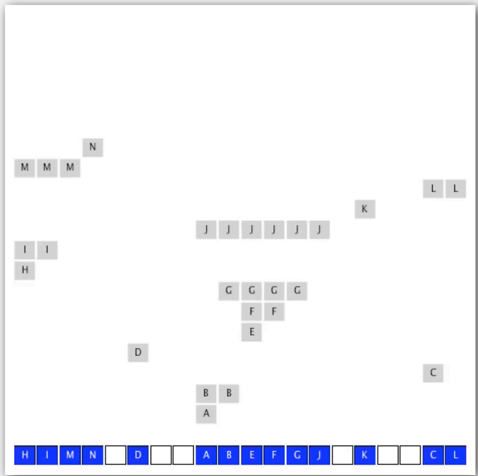
Linear probing ST implementation

```
public class LinearProbingHashST<Key, Value>
{
   private int M = 30001;
   private Value[] vals = (Value[]) new Object[M];
   private Key[]   keys = (Key[])   new Object[M];

   private int hash(Key key)    { /* as before      */  }

   private Value get(Key key)   { /* previous slide */  }

   public void put(Key key, Value val)
   {
      int i;
      for (i = hash(key); keys[i] != null; i = (i+1) % M)
         if (keys[i].equals(key))
             break;
      keys[i] = key;
      vals[i] = val;
   }

}
```

Clustering

Cluster. A contiguous block of items.

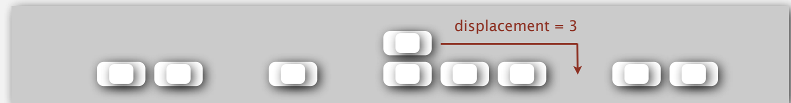
Observation. New keys likely to hash into middle of big clusters.



Knuth's parking problem

Model. Cars arrive at one-way street with M parking spaces. Each desires a random space i : if space i is taken, try $i+1$, $i+2$, etc.

Q. What is mean displacement of a car?



Half-full. With $M/2$ cars, mean displacement is $\sim 3/2$.

Full. With M cars, mean displacement is $\sim \sqrt{\pi M/8}$.

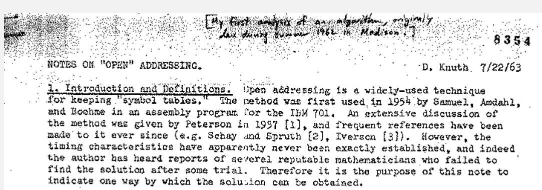
Analysis of linear probing

Proposition. Under uniform hashing assumption, the average # of probes in a linear probing hash table of size M that contains $N = \alpha M$ keys is:

$$\sim \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right) \qquad \sim \frac{1}{2}\left(1 + \frac{1}{(1-\alpha)^2}\right)$$

search hit — search miss / insert

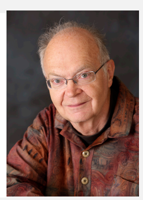
Pf.


[My first analysis of an algorithm, originally done during summer 1962 in Madison.]

8354

NOTES ON "OPEN" ADDRESSING. D. Knuth 7/22/63

1. Introduction and Definitions. Open addressing is a widely-used technique for keeping "symbol tables." The method was first used in 1954 by Samuel, Amdahl, and Boehme in an assembly program for the IBM 701. An extensive discussion of the method was given by Peterson in 1957 [1], and frequent references have been made to it ever since (e.g. Schay and Spruth [2], Iverson [3]). However, the timing characteristics have apparently never been exactly established, and indeed the author has heard reports of several reputable mathematicians who failed to find the solution after some trial. Therefore it is the purpose of this note to indicate one way by which the solution can be obtained.



Parameters.

- M too large $\Rightarrow$ too many empty array entries.
- M too small $\Rightarrow$ search time blows up.
- Typical choice: $\alpha = N/M \sim ½$. ← # probes for search hit is about 3/2; # probes for search miss is about 5/2

ST implementations: summary

implementation	worst-case cost (after N inserts)			average case (after N random inserts)			ordered iteration?	key interface
	search	insert	delete	search hit	insert	delete		
sequential search (unordered list)	N	N	N	N/2	N	N/2	no	equals()
binary search (ordered array)	lg N	N	N	lg N	N/2	N/2	yes	compareTo()
BST	N	N	N	1.38 lg N	1.38 lg N	?	yes	compareTo()
red-black tree	2 lg N	2 lg N	2 lg N	1.00 lg N	1.00 lg N	1.00 lg N	yes	compareTo()
separate chaining	lg N *	lg N *	lg N *	3-5 *	3-5 *	3-5 *	no	equals() hashCode()
linear probing	lg N *	lg N *	lg N *	3-5 *	3-5 *	3-5 *	no	equals() hashCode()

* under uniform hashing assumption

http://algs4.cs.princeton.edu

3.4 Hash Tables

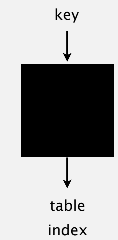
- basic ideas
- separate chaining
- linear probing
- **hash functions**
- context

Computing the hash function

Idealistic goal. Scramble the keys uniformly to produce a table index.

- Efficiently computable.
- Each table index equally likely for each key.

thoroughly researched problem, still problematic in practical applications



Ex 1. Phone numbers.

- Bad: first three digits.
- Better: last three digits.

Ex 2. Social Security numbers.

- Bad: first three digits.
- Better: last three digits.

573 = California, 574 = Alaska (assigned in chronological order within geographic region)

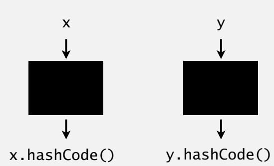
Practical challenge. Need different approach for each key type.

Java's hash code conventions

All Java classes inherit a method `hashCode()`, which returns a 32-bit `int`.

Requirement. If `x.equals(y)`, then `(x.hashCode() == y.hashCode())`.

Highly desirable. If `!x.equals(y)`, then `(x.hashCode() != y.hashCode())`.



Default implementation. Memory address of `x`.
Legal (but poor) implementation. Always return `17`.
Customized implementations. `Integer`, `Double`, `String`, `File`, `URL`, `Date`, …
User-defined types. Users are on their own.

Implementing hash code: integers, booleans, and doubles

Java library implementations

```
public final class Integer
{
   private final int value;
   ...

   public int hashCode()
   {  return value;  }
}
```

```
public final class Boolean
{
   private final boolean value;
   ...

   public int hashCode()
   {
      if (value) return 1231;
      else       return 1237;
   }
}
```

```
public final class Double
{
   private final double value;
   ...

   public int hashCode()
   {
      long bits = doubleToLongBits(value);
      return (int) (bits ^ (bits >>> 32));
   }
}
```

convert to IEEE 64-bit representation; xor most significant 32-bits with least significant 32-bits

Implementing hash code: strings

Java library implementation

```
public final class String
{
   private final char[] s;
   ...

   public int hashCode()
   {
      int hash = 0;
      for (int i = 0; i < length(); i++)
         hash = s[i] + (31 * hash);
      return hash;
   }
}
```

i^{th} character of s

char	Unicode
…	…
'a'	97
'b'	98
'c'	99
…	…

- Horner's method to hash string of length L: L multiplies/adds.
- Equivalent to $h = s[0] \cdot 31^{L-1} + \ldots + s[L-3] \cdot 31^2 + s[L-2] \cdot 31^1 + s[L-1] \cdot 31^0$.

Ex.

```
String s = "call";
int code = s.hashCode();
```

$3045982 = 99 \cdot 31^3 + 97 \cdot 31^2 + 108 \cdot 31^1 + 108 \cdot 31^0$
$= 108 + 31 \cdot (108 + 31 \cdot (97 + 31 \cdot (99)))$
(Horner's method)

Implementing hash code: strings

Performance optimization.

- Cache the hash value in an instance variable.
- Return cached value.

```
public final class String
{
   private int hash = 0;
   private final char[] s;
   ...

   public int hashCode()
   {
      int h = hash;
      if (h != 0) return h;
      for (int i = 0; i < length(); i++)
         h = s[i] + (31 * hash);
      hash = h;
      return h;
   }
}
```

cache of hash code
return cached value
store cache of hash code

Implementing hash code: user-defined types

```
public final class Transaction implements Comparable<Transaction>
{
   private final String  who;
   private final Date    when;
   private final double  amount;

   public Transaction(String who, Date when, double amount)
   {  /* as before */  }

   ...

   public boolean equals(Object y)
   {  /* as before */  }

   public int hashCode()
   {
      int hash = 17;
      hash = 31*hash + who.hashCode();
      hash = 31*hash + when.hashCode();
      hash = 31*hash + ((Double) amount).hashCode();
      return hash;
   }
}
```

nonzero constant
for reference types, use hashCode()
for primitive types, use hashCode() of wrapper type
typically a small prime

Hash code design

"Standard" recipe for user-defined types.

- Combine each significant field using the $31x + y$ rule.
- If field is a primitive type, use wrapper type `hashCode()`.
- If field is null, return 0.
- If field is a reference type, use `hashCode()`. ← applies rule recursively
- If field is an array, apply to each entry. ← or use `Arrays.deepHashCode()`

In practice. Recipe works reasonably well; used in Java libraries.
In theory. Keys are bitstring; "universal" hash functions exist.

Basic rule. Need to use the whole key to compute hash code; consult an expert for state-of-the-art hash codes.

http://algs4.cs.princeton.edu

3.4 Hash Tables

- basic ideas
- separate chaining
- linear probing
- hash functions
- context

War story: String hashing in Java

String `hashCode()` in Java 1.1.

- For long strings: only examine 8-9 evenly spaced characters.
- Benefit: saves time in performing arithmetic.

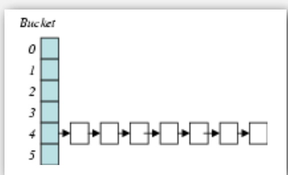
```
public int hashCode()
{
   int hash = 0;
   int skip = Math.max(1, length() / 8);
   for (int i = 0; i < length(); i += skip)
      hash = s[i] + (37 * hash);
   return hash;
}
```

- Downside: great potential for bad collision patterns.

```
http://www.cs.princeton.edu/introcs/13loop/Hello.java
http://www.cs.princeton.edu/introcs/13loop/Hello.class
http://www.cs.princeton.edu/introcs/13loop/Hello.html
http://www.cs.princeton.edu/introcs/12type/index.html
↑     ↑     ↑     ↑     ↑     ↑     ↑     ↑
```


War story: algorithmic complexity attacks

Q. Is the uniform hashing assumption important in practice?
A. Obvious situations: aircraft control, nuclear reactor, pacemaker.
A. Surprising situations: denial-of-service attacks.



malicious adversary learns your hash function (e.g., by reading Java API) and causes a big pile-up in single slot that grinds performance to a halt

Real-world exploits. [Crosby-Wallach 2003]

- Bro server: send carefully chosen packets to DOS the server, using less bandwidth than a dial-up modem.
- Perl 5.8.0: insert carefully chosen strings into associative array.
- Linux 2.4.20 kernel: save files with carefully chosen names.

Algorithmic complexity attack on Java

Goal. Find family of strings with the same hash code.
Solution. The base 31 hash code is part of Java's string API.

key	hashCode()
"Aa"	2112
"BB"	2112

key	hashCode()
"AaAaAaAa"	-540425984
"AaAaAaBB"	-540425984
"AaAaBBAa"	-540425984
"AaAaBBBB"	-540425984
"AaBBAaAa"	-540425984
"AaBBAaBB"	-540425984
"AaBBBBAa"	-540425984
"AaBBBBBB"	-540425984

key	hashCode()
"BBAaAaAa"	-540425984
"BBAaAaBB"	-540425984
"BBAaBBAa"	-540425984
"BBAaBBBB"	-540425984
"BBBBAaAa"	-540425984
"BBBBAaBB"	-540425984
"BBBBBBAa"	-540425984
"BBBBBBBB"	-540425984

2^N strings of length 2N that hash to same value!

Diversion: one-way hash functions

One-way hash function. "Hard" to find a key that will hash to a desired value (or two keys that hash to same value).

Ex. MD4, MD5, SHA-0, SHA-1, SHA-2, WHIRLPOOL, RIPEMD-160, ….

known to be insecure

```
String password = args[0];
MessageDigest sha1 = MessageDigest.getInstance("SHA1");
byte[] bytes = sha1.digest(password);

/* prints bytes as hex string */
```

Applications. Digital fingerprint, message digest, storing passwords.
Caveat. Too expensive for use in ST implementations.

Separate chaining vs. linear probing

Separate chaining.
- Easier to implement delete.
- Performance degrades gracefully.
- Clustering less sensitive to poorly-designed hash function.

Linear probing.
- Less wasted space.
- Better cache performance.

Q. How to delete from linear probing?
Q. How to resize from linear probing?

Hashing: variations on the theme

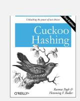
Many improved versions have been studied.

Two-probe hashing. (separate-chaining variant)
- Hash to two positions, insert key in shorter of the two chains.
- Reduces expected length of the longest chain to $\log \log N$.

Double hashing. (linear-probing variant)
- Use linear probing, but skip a variable amount, not just 1 each time.
- Effectively eliminates clustering.
- Can allow table to become nearly full.
- More difficult to implement delete.

Based on second hash function

Cuckoo hashing. (linear-probing variant)
- Hash key to two positions; insert key into either position; if occupied, reinsert displaced key into its alternative position (and recur).
- Constant worst case time for search.



Hash tables vs. balanced search trees

Hash tables.
- Simpler to code.
- No effective alternative for unordered keys.
- Faster for simple keys (a few arithmetic ops versus $\log N$ compares).
- Better system support in Java for strings (e.g., cached hash code).

Balanced search trees.
- Stronger performance guarantee.
- Support for ordered ST operations.
- Easier to implement `compareTo()` correctly than `equals()` and `hashCode()`.

Java system includes both.
- Red-black BSTs: `java.util.TreeMap`, `java.util.TreeSet`.
- Hash tables: `java.util.HashMap`, `java.util.IdentityHashMap`.