



<http://algs4.cs.princeton.edu>

## 3.4 HASH TABLES

---

- ▶ *hash functions*
- ▶ *separate chaining*
- ▶ *linear probing*
- ▶ ***context***

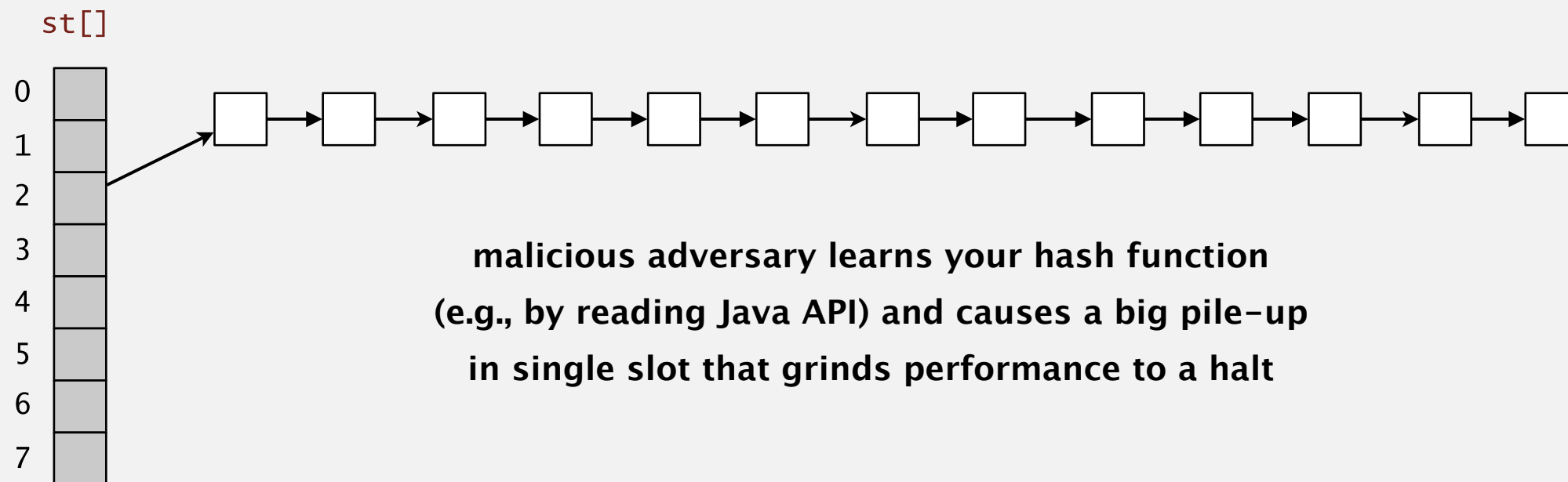
# War story: algorithmic complexity attacks

---

Q. Is the uniform hashing assumption important in practice?

A. Obvious situations: aircraft control, nuclear reactor, pacemaker.

A. Surprising situations: **denial-of-service** attacks.



**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.

# War story: algorithmic complexity attacks

---

## A Java bug report.

Jan Lieskovsky 2011-11-01 10:13:47 EDT

Description

Julian Wälde and Alexander Klink reported that the `String.hashCode()` hash function is not sufficiently collision resistant. `hashCode()` value is used in the implementations of `HashMap` and `Hashtable` classes:

<http://docs.oracle.com/javase/6/docs/api/java/util/HashMap.html>

<http://docs.oracle.com/javase/6/docs/api/java/util/Hashtable.html>

A specially-crafted set of keys could trigger hash function collisions, which can degrade performance of `HashMap` or `Hashtable` by changing hash table operations complexity from an expected/average  $O(1)$  to the worst case  $O(n)$ . Reporters were able to find colliding strings efficiently using equivalent substrings and meet in the middle techniques.

This problem can be used to start a denial of service attack against Java applications that use untrusted inputs as `HashMap` or `Hashtable` keys. An example of such application is web application server (such as tomcat, see [bug #750521](#)) that may fill hash tables with data from HTTP request (such as GET or POST parameters). A remote attack could use that to make JVM use excessive amount of CPU time by sending a POST request with large amount of parameters which hash to the same value.

This problem is similar to the issue that was previously reported for and fixed in e.g. perl:

[http://www.cs.rice.edu/~scrosby/hash/CrosbyWallach\\_UsenixSec2003.pdf](http://www.cs.rice.edu/~scrosby/hash/CrosbyWallach_UsenixSec2003.pdf)

# 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
"BBBBBBaAa"	-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.

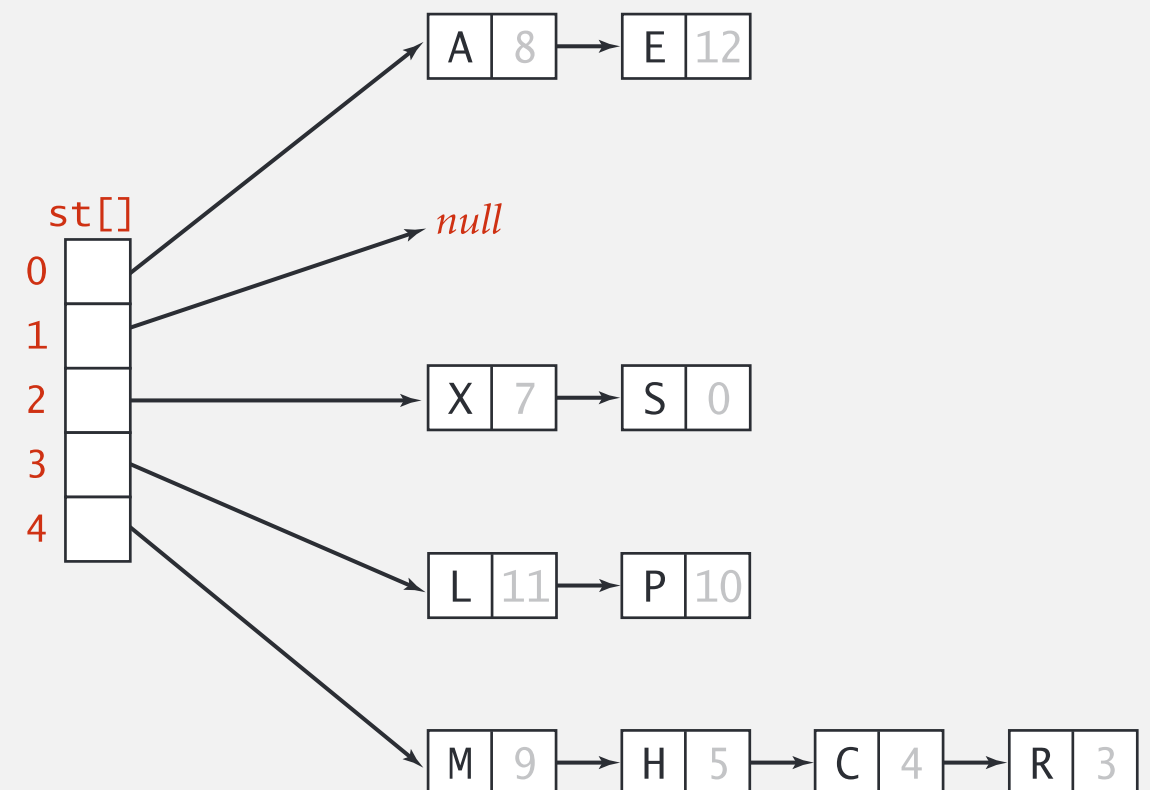
# Separate chaining vs. linear probing

## Separate chaining.

- Performance degrades gracefully.
- Clustering less sensitive to poorly-designed hash function.

## Linear probing.

- Less wasted space.
- Better cache performance.



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

# 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.

## 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`.