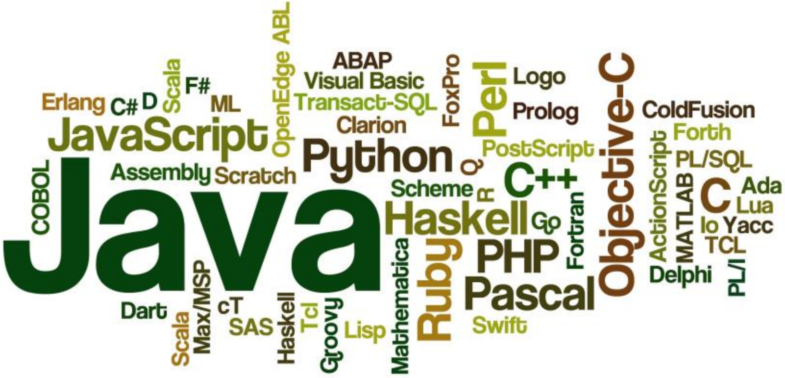
Introduction:

Hash Tables

Data Structures and Algorithms

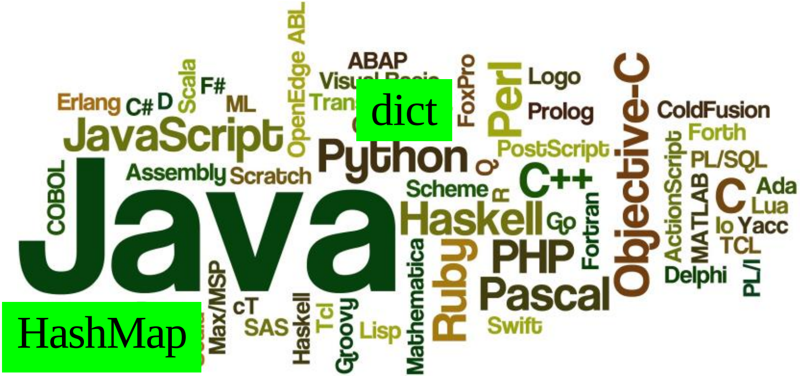
Outline

Programming Languages



Programming Languages

Programming Languages



Programming Languages

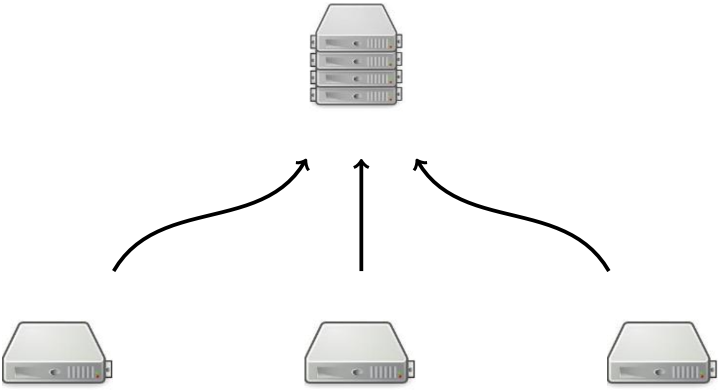
dict

HashMap

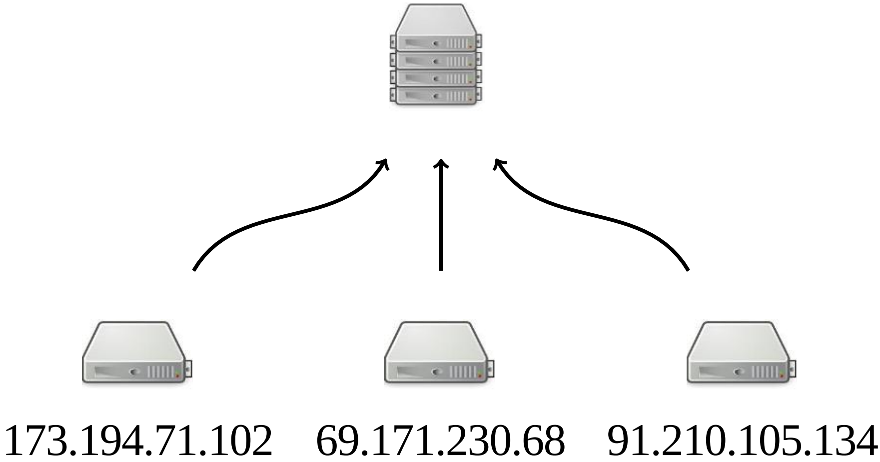
for, if, while, int

Outline

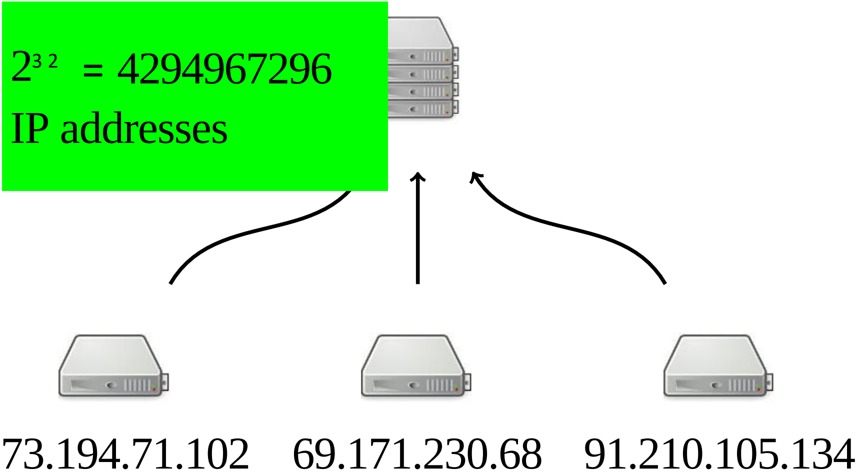
Web Service



Web Service



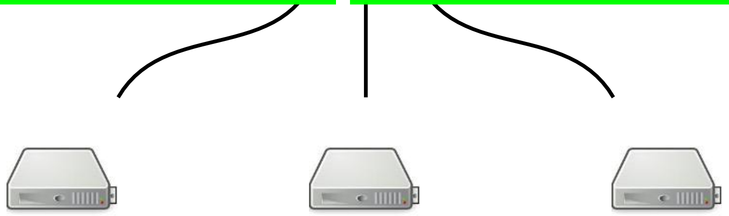
Web Service


$2^{32} = 4294967296$
IP addresses
173.194.71.102
69.171.230.68
91.210.105.134

Web Service

$2^{32} = 4294967296$
IP addresses

2^{128} IPv6 addresses
number with 39 digits!



Access Log

Date	Time	IP address
09 Dec 2015	00:45:13	173.194.71.102
09 Dec 2015	00:45:15	69.171.230.68
...	...	...
...	...	...
09 Dec 2015	01:45:13	91.210.105.134

IP Access List

Analyse the access log and quickly answer queries: did anybody access the service from this *IP* during the last hour? How many times? How many *IP*s were used to access the service during the last hour?

Log Processing

- 1h of logs can contain millions of lines

Log Processing

- 1h of logs can contain millions of lines
- Too slow to process that for each query

Log Processing

- 1h of logs can contain millions of lines
- Too slow to process that for each query
- Keep count: how many times each IP appears in the last 1h of the access log

Log Processing

- 1h of logs can contain millions of lines
- Too slow to process that for each query
- Keep count: how many times each IP appears in the last 1h of the access log
- C is some data structure to store the mapping from IPs to counters

Log Processing

- 1h of logs can contain millions of lines
- Too slow to process that for each query
- Keep count: how many times each IP appears in the last 1h of the access log
- C is some data structure to store the mapping from IPs to counters
- We will learn later how to implement C

Log Processing

Time	IP address
00:45:13	173.194.71.102
00:45:13	69.171.230.68
...	...
01:45:13	173.194.71.102
01:45:13	91.210.105.134

Log Processing

Now

Time	IP address
00:45:13	173.194.71.102
00:45:13	69.171.230.68
...	...
01:45:13	173.194.71.102
01:45:13	91.210.105.134

Log Processing

Time	IP address
00:45:13	173.194.71.102
00:45:13	69.171.230.68
...	...
01:45:13	173.194.71.102
01:45:13	91.210.105.134

Now

Increment counter

Log Processing

1 hour ago

Now

Time	IP address
00:45:13	173.194.71.102
00:45:13	69.171.230.68
...	...
01:45:13	173.194.71.102
01:45:13	91.210.105.134

Increment counter

Log Processing

Decrement counter

1 hour ago

Time	IP address
00:45:13	173.194.71.102
00:45:13	69.171.230.68
...	...
01:45:13	173.194.71.102
01:45:13	91.210.105.134

Now

Increment counter

Coming Next

How to implement the mapping C ?

Outline

Direct Addressing

- Need a data structure for C

Direct Addressing

- Need a data structure for C
- There are 2^{32} different IP(v4) addresses

Direct Addressing

- Need a data structure for *C*
- There are 2^{32} different IP(v4)
- addresses Convert IP to 32-bit integer

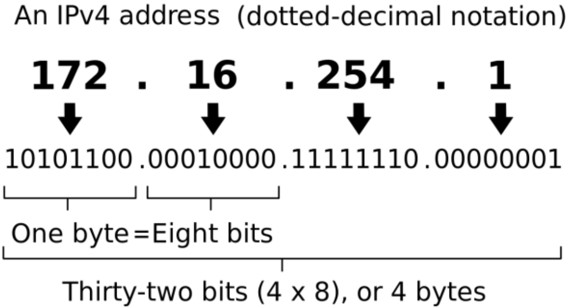
Direct Addressing

- Need a data structure for *C*
- There are 2^{32} di erent IP(v4) addresses
- Convert IP to 32-bit integer
- Create an integer array *A* of size 2^{32}

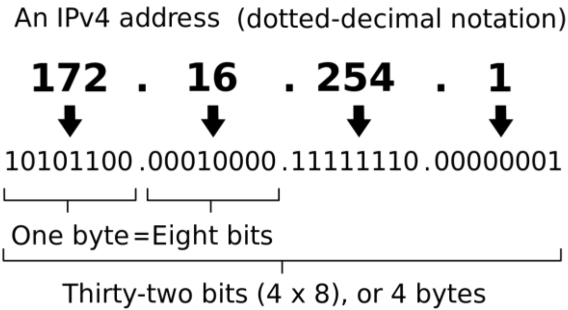
Direct Addressing

- Need a data structure for C
- There are 2^{32} di erent IP(v4) addresses
- Convert IP to 32-bit integer
- Create an integer array A of size 2^{32}
- Use $A[\mathrm{int}(IP)]$ as $C[IP]$

int(IP)

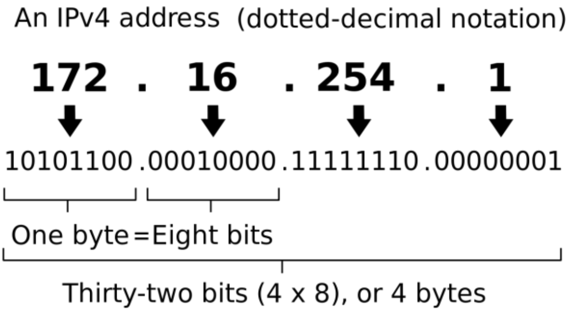

An IPv4 address (dotted-decimal notation)
172 . 16 . 254 . 1
10101100.00010000.11111110.00000001
One byte = Eight bits
Thirty-two bits (4 x 8), or 4 bytes

int(IP)



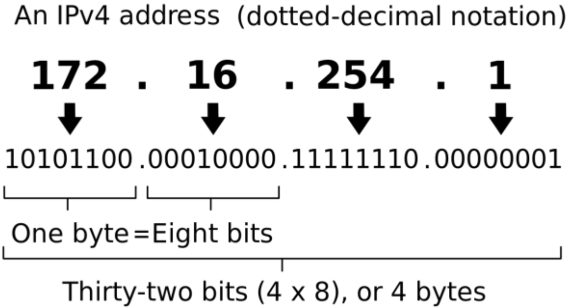
- int(0.0.0.1) = 1

int(IP)



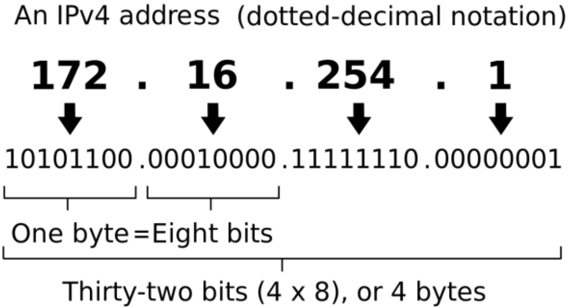
- int(0.0.0.1) = 1
- int(172.16.254.1) = 2886794753

int(IP)



- int(0.0.0.1) = 1
- int(172.16.254.1) = 2886794753
- int(69.171.230.68) =

int(IP)



- int(0.0.0.1) = 1
- int(172.16.254.1) = 2886794753
- int(69.171.230.68) = 1168893508

Asymptotics

- `UpdateAccessList` is $O(1)$ per log line

Asymptotics

- `UpdateAccessList` is $O(1)$ per log line
- `AccessedLastHour` is $O(1)$

Asymptotics

- UpdateAccessList is $O(1)$ per log line
- AccessedLastHour is $O(1)$
- But need 2^{32} memory even for few IPs

Asymptotics

- UpdateAccessList is $O(1)$ per log line
- AccessedLastHour is $O(1)$
- But need 2^{32} memory even for few IPs
- IPv6: 2^{128} won't fit in memory

Asymptotics

- UpdateAccessList is $O(1)$ per log line
- AccessedLastHour is $O(1)$
- But need 2^{32} memory even for few IPs
- IPv6: 2^{128} won't fit in memory
- In general: $O(N)$ memory, $N = |S|$

Outline

Encoding IPs

- Encode IPs with small numbers

Encoding IPs

- Encode IPs with small numbers
- I.e. numbers from 0 to 999

Encoding IPs

- Encode IPs with small numbers
- i.e. numbers from 0 to 999
- Different codes for currently active IPs

Hash Function

Definition

For any set of objects S and any integer $m > 0$, a function $h : S \to \{0, 1, \ldots, m-1\}$ is called a hash function.

Hash Function

Definition

For any set of objects S and any integer $m > 0$, a function $h : S \longrightarrow \{0, 1, \ldots, m-1\}$ is called a hash function.

Definition

m is called the cardinality of hash function h.

Desirable Properties

- h should be fast to compute

Desirable Properties

- *h* should be fast to compute
- Different values for different objects

Desirable Properties

- h should be fast to compute
- Different values for different objects
- Direct addressing with $O(m)$ memory

Desirable Properties

- h should be fast to compute
- Different values for different objects
- Direct addressing with $O(m)$ memory
- Want small cardinality m

Desirable Properties

- h should be fast to compute
- Di erent values for di erent objects
- Direct addressing with $O(m)$ memory
- Want small cardinality m
- Impossible to have all different values if number of objects $|S|$ is more than m

Popular Hash Function

Division method: Choose a number m larger than the number n of keys in K. (The number m is usually chosen to be a prime number or a number without small divisors, since this frequently minimizes the number of collisions.) The hash function H is defined by

$H(k) = k \pmod{m}$ or $H(k) = k \pmod{m} + 1$

Here $k \pmod{m}$ denotes the remainder when k is divided by m. The second formula is used when we want the hash addresses to range from 1 to m rather than from 0 to $m – 1$.

Popular Hash Function

Midsquare method: The key k is squared. Then the hash function H is defined by

H(k) = l

where l is obtained by deleting digits from both ends of k^2. We emphasize that the same positions of k^2 must be used for all of the keys.

Popular Hash Function

Folding method: The key k is partitioned into a number of parts, k1, ...,kr , where each part, except possibly the last, has the same number of digits as the required address. Then the parts are added together, ignoring the last carry. That is,

$H(k) = k_1 + k_2 + ... + k_r$

where the leading-digit carries, if any, are ignored. Sometimes, for extra “milling,” the even-numbered parts, k_2, k_4, ..., are each reversed before the addition.

Question

Suppose a company with 68 employees assigns a 4-digit employee number to each employee which is used as the primary key in the company's employee file. Suppose L consists of 100 two-digit addresses: 00, 01, 02, ..., 99. Compute the locations to which the keys 3205, 7148, and 2345 are mapped.

(a) *Division method*. Choose a prime number m close to 99, such as $m = 97$. Then

$$H(3205) = 4,\ H(7148) = 67,\ H(2345) = 17$$

That is, dividing 3205 by 97 gives a remainder of 4, dividing 7148 by 97 gives a remainder of 67, and dividing 2345 by 97 gives a remainder of 17. In the case that the memory addresses begin with 01 rather than 00, we choose that the function $H(k) = k(\bmod m) + 1$ to obtain:

$$H(3205) = 4 + 1 = 5,\ H(7148) = 67 + 1 = 68,\ H(2345) = 17 + 1 = 18$$

(b) *Midsquare method*. The following calculations are performed:

k:	3205	7148	2345
k^2:	10 272 025	51 093 904	5 499 025
$H(k)$:	72	93	99

Observe that the fourth and fifth digits, counting from the right, are chosen for the hash address.

Question

Suppose a company with 68 employees assigns a 4-digit employee number to each employee which is used as the primary key in the company's employee file. Suppose L consists of 100 two-digit addresses: 00, 01, 02, ..., 99. Compute the locations to which the keys 3205, 7148, and 2345 are mapped.

(c) *Folding method*. Chopping the key k into two parts and adding yields the following hash addresses:

$H(3205) = 32 + 05 = 37$, $H(7148) = 71 + 48 = 19$, $H(2345) = 23 + 45 = 68$

Observe that the leading digit 1 in $H(7148)$ is ignored. Alternatively, one may want to reverse the second part before adding, thus producing the following hash addresses:

$H(3205) = 32 + 50 = 82$, $H(7148) = 71 + 84 + 55$, $H(2345) = 23 + 54 = 77$

Question

Consider a hash table of size m=1000 and a corresponding hash function h(k)=⌊m(kAmod1)⌋ for A=(√5−1)/2. Compute the locations to which the keys 6161, 6262, 6363, 6464, and 6565 are mapped.

- $h(61) = \lfloor 1000(61 \cdot \frac{\sqrt{5}-1}{2} \mod 1) \rfloor = 700.$
- $h(62) = \lfloor 1000(62 \cdot \frac{\sqrt{5}-1}{2} \mod 1) \rfloor = 318.$
- $h(63) = \lfloor 1000(63 \cdot \frac{\sqrt{5}-1}{2} \mod 1) \rfloor = 936.$
- $h(64) = \lfloor 1000(64 \cdot \frac{\sqrt{5}-1}{2} \mod 1) \rfloor = 554.$
- $h(65) = \lfloor 1000(65 \cdot \frac{\sqrt{5}-1}{2} \mod 1) \rfloor = 172.$

Collisions

Definition

When $h(o_1) = h(o_2)$ and $o_1 \neq o_2$, this is a collision.

Outline

Map

Store mapping from objects to other objects:

- Filename → location of the file on disk
- Student ID → student name
- Contact name → contact phone number

Map

Store mapping from objects to other objects:

- Filename $\rightarrow$ location of the le on disk
- Student ID $\rightarrow$ student name
- Contact name $\rightarrow$ contact phone number

Definition

Map from S to V is a data structure with methods HasKey(O), Get(O), Set(O, v), where $O \in S$, $v \in V$.

Chaining

0
1
2
3
4
5
6
7

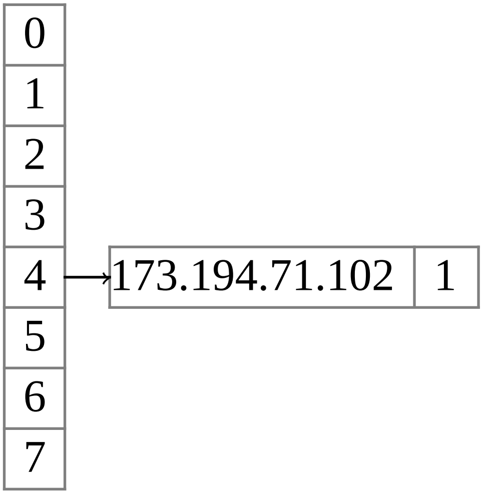
Chaining

$$h(173.194.71.102) = 4$$

0
1
2
3
4
5
6
7

Chaining

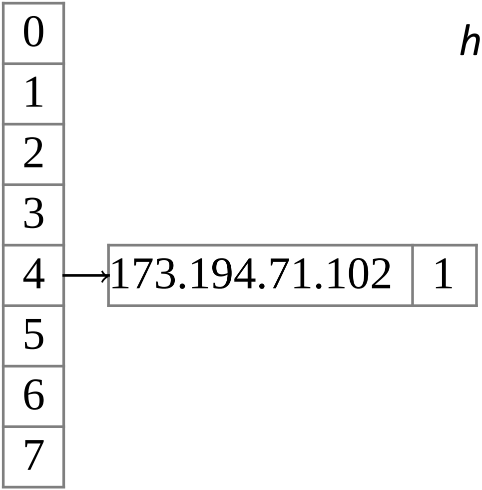
$$h(173.194.71.102) = 4$$



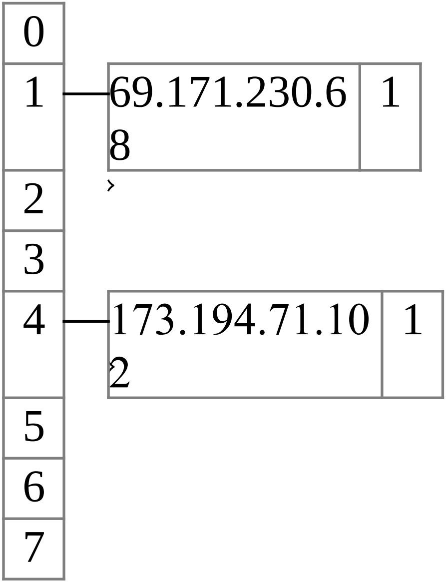
Chaining

$$h(173.194.71.102) = 4$$

$$h(69.171.230.68) = 1$$



Chaining



$$h(173.194.71.102) = 4$$

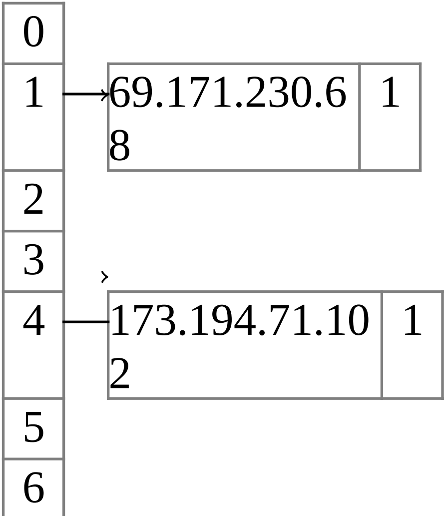
$$h(69.171.230.68) = 1$$

Chaining

$$h(173.194.71.102) = 4$$

$$h(69.171.230.68) = 1$$

$$h(173.194.71.102) = 4$$

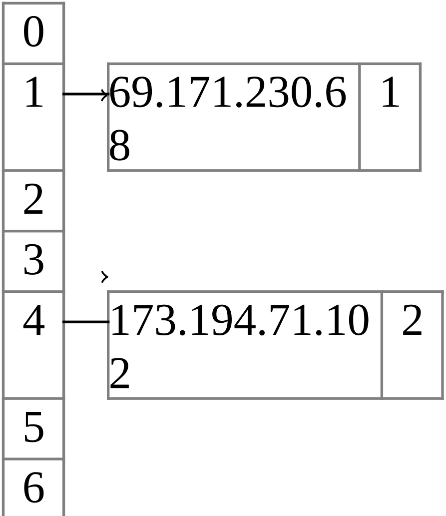


Chaining

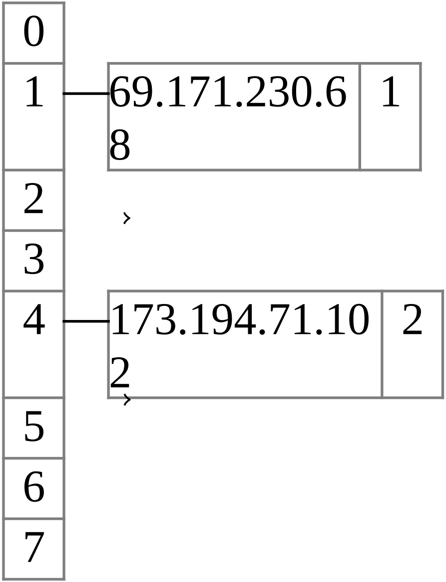
$h(173.194.71.102) = 4$

$h(69.171.230.68) = 1$

$h(173.194.71.102) = 4$



Chaining



$h(173.194.71.102) = 4$

$h(69.171.230.68) = 1$

$h(173.194.71.102) = 4$

$h(91.210.105.134) = 4$

Chaining

$$h(173.194.71.102) = 4$$

$$h(69.171.230.68) = 1$$

$$h(173.194.71.102) = 4$$

$$h(91.210.105.134) = 4$$

0

1 69.171.230.68 1

2 →

3

4 → 173.194.71.102 | 2 → 91.210.105.134 | 1

5

6

7

Chains — array of chains

Each chain is a list of pairs (object, value)

HasKey(object)

```
chain ← Chains[hash(object)]
for (key, value) in chain:
  if key == object:
    return true
return false
```

Get(object)

```
chain ← Chains[hash(object)]
for (key, value) in chain:
  if key == object:
    return value
return N/A
```

Set(object, value)

```
chain ← Chains[hash(object)]
for pair in chain:
  if pair.key == object:
    pair.value ← value
    return
chain.Append((object, value))
```

Lemma

Let c be the length of the longest chain in *chains*. Then the running time of HasKey, Get, Set is $\Theta(c+1)$.

Lemma

Let c be the length of the longest chain in A. Then the running time of HasKey, Get, Set is $\Theta(c+1)$.

Proof

- If $L = A[h(O)], \mathrm{len}(L) = c, O \notin L$, need to scan all c items

Lemma

Let c be the length of the longest chain in A. Then the running time of HasKey, Get, Set is $\Theta(c+1)$.

Proof

- If $L = A[h(O)], \mathrm{len}(L) = c, O \notin L$, need to scan all c items
- If $c = 0$, we still need $O(1)$ time $\square$

Lemma

Let n be the number of di erent keys O currently in the map and m be the cardinality of the hash function. Then the memory consumption for chaining is $\Theta(n + m)$.

Lemma

Let n be the number of different keys O currently in the map and m be the cardinality of the hash function. Then the memory consumption for chaining is $\Theta(n+m)$.

Proof

- $\Theta(n)$ to store n pairs (O, v)

Lemma

Let n be the number of different keys O currently in the map and m be the cardinality of the hash function. Then the memory consumption for chaining is $\Theta(n + m)$.

Proof

- $\Theta(n)$ to store n pairs (O, v)
- $\Theta(m)$ to store array A of size m □

Outline

Set

Definition

Set is a data structure with methods $\text{Add}(O)$, $\text{Remove}(O)$, $\mathtt{Find}(O)$.

Set

Definition

Set is a data structure with methods Add(O), Remove(O), $\texttt{Find}(O)$.

Examples

- IPs accessed during last hour

Set

Definition

Set is a data structure with methods $\text{Add}(O)$, $\text{Remove}(O)$, $\texttt{Find}(O)$.

Examples

- IPs accessed during last hour
- Students on campus

Set

Definition

Set is a data structure with methods
Add(O), Remove(O), Find(O).

Examples

- IPs accessed during last hour
- Students on campus
- Keywords in a programming language

Implementing Set

Two ways to implement a set using chaining:

- Set is equivalent to map from S to $V = \{true, false\}$

Implementing Set

Two ways to implement a set using chaining:

- Set is equivalent to map from S to $V = \{true, false\}$
- Store just objects O instead of pairs (O, v) in chains

$h : S \rightarrow \{0, 1, \ldots, m-1\}$

$O,\ O' \in S$

$A \leftarrow$ array of m lists (chains) of objects O

Find(O)

```
L ← A[h(O)]
for O′ in L:
   if O′ == O:
      return true
return false
```

Add(O)

```
L ← A[h(O)]
for O′ in L:
    if O′ == O:
        return
L.Append(O)
```

Remove(O)

```
if not Find(O):
    return
L ← A[h(O)]
L.Erase(O)
```

Hash Table

Definition

An implementation of a set or a map using hashing is called a hash table.

Programming Languages

Set:

- `unordered_set` in C++
- `HashSet` in Java
- `set` in Python

Map:

- `unordered_map` in C++
- HashMap in Java
- `dict` in Python

Conclusion

- Chaining is a technique to implement a hash table

Conclusion

- Chaining is a technique to implement a hash table
- Memory consumption is $O(n+m)$

Conclusion

- Chaining is a technique to implement a hash table
- Memory consumption is $O(n+m)$
- Operations work in time $O(c+1)$

Conclusion

- Chaining is a technique to implement a hash table
- Memory consumption is $O(n + m)$
- Operations work in time $O(c + 1)$
- How to make both m and c small?