Hash Tables: Hash Functions

Michael Levin

Higher School of Economics

Data Structures
Data Structures and Algorithms

Outline

Phone Book

Design a data structure to store your contacts: names of people along with their phone numbers. The data structure should be able to do the following quickly:

- Add and delete contacts,
- Lookup the phone number by name,
- Determine who is calling given their phone number.

- We need two Maps:
  (phone number $\rightarrow$ name) and
  (name $\rightarrow$ phone number)

- We need two Maps:
  (phone number $\rightarrow$ name) and
  (name $\rightarrow$ phone number)
- Implement these Maps as hash tables

- We need two Maps:
  (phone number $\rightarrow$ name) and
  (name $\rightarrow$ phone number)
- Implement these Maps as hash tables
- First, we will focus on the Map from phone numbers to names

Direct Addressing

- $\texttt{int}(\texttt{123-45-67}) = 1234567$

Direct Addressing

- $\texttt{int}(123\text{-}45\text{-}67) = 1234567$
- Create array *Name* of size 10^L where L is the maximum allowed phone number length

Direct Addressing

- $\texttt{int}(123\text{-}45\text{-}67) = 1234567$
- Create array *Name* of size 10^L where L is the maximum allowed phone number length
- Store the name corresponding to phone number P in $Name[\texttt{int}(P)]$

Direct Addressing

- $\texttt{int}(\texttt{123-45-67}) = 1234567$
- Create array *Name* of size 10^L where L is the maximum allowed phone number length
- Store the name corresponding to phone number P in $Name[\texttt{int}(P)]$
- If no contact with phone number P, $Name[\texttt{int}(P)] = \texttt{N/A}$

Direct Addressing

Natalie: 123-45-67 $\longrightarrow$ 1234567

Steve: 223-23-23 $\longrightarrow$ 2232323

Name
...
Natalie
N/A
N/A
...
Steve
N/A
...

Direct Addressing

- Operations run in $O(1)$

Direct Addressing

- Operations run in $O(1)$
- Memory usage: $O(10^L)$, where L is the maximum length of a phone number

Direct Addressing

- Operations run in $O(1)$
- Memory usage: $O(10^L)$, where L is the maximum length of a phone number
- Problematic with international numbers of length 12 and more: we will need 10^{12} bytes $=$ 1TB to store one person's phone book — this won't fit in anyone's phone!

Chaining

- Select hash function h with cardinality m

Chaining

- Select hash function h with cardinality m
- Create array *Name* of size m

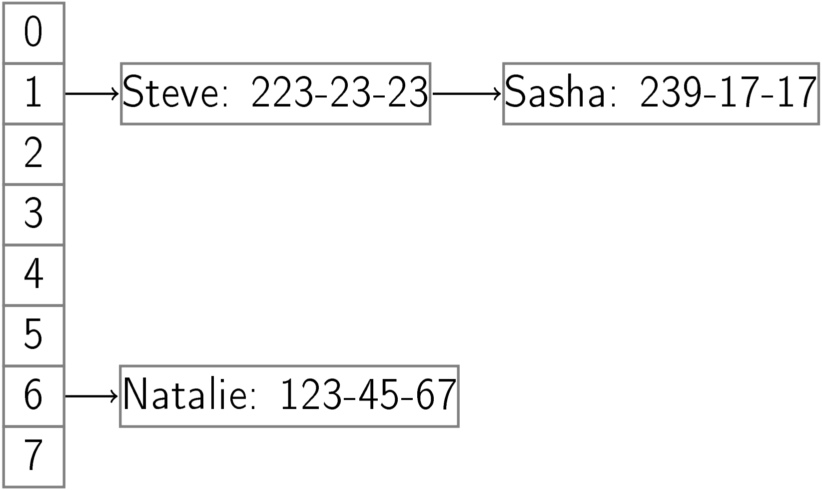
Chaining

- Select hash function h with cardinality m
- Create array $Name$ of size m
- Store chains in each cell of the array $Name$

Chaining

- Select hash function h with cardinality m
- Create array *Name* of size m
- Store chains in each cell of the array *Name*
- Chain $Name[h(\texttt{int}(P))]$ contains the name for phone number P

Chaining



Parameters

- n phone numbers stored

Parameters

- n phone numbers stored
- m — cardinality of the hash function

Parameters

- n phone numbers stored
- m — cardinality of the hash function
- c — length of the longest chain

Parameters

- n phone numbers stored
- m — cardinality of the hash function
- c — length of the longest chain
- $O(n + m)$ memory is used

Parameters

- n phone numbers stored
- m — cardinality of the hash function
- c — length of the longest chain
- $O(n+m)$ memory is used
- $\alpha = \frac{n}{m}$ is called load factor

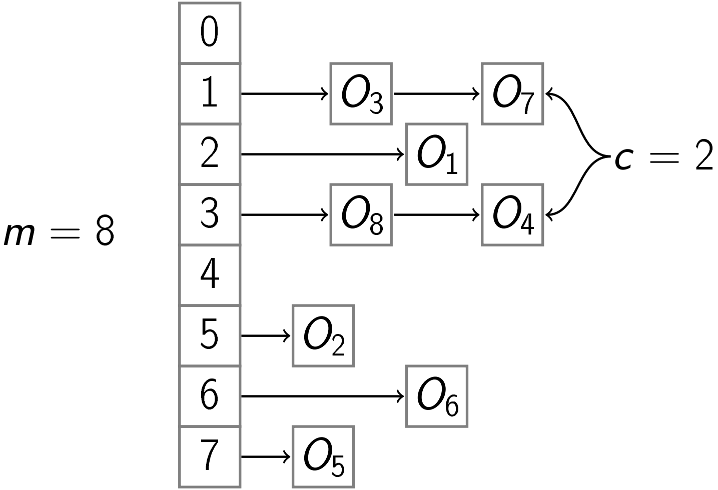
Parameters

- n phone numbers stored
- m — cardinality of the hash function
- c — length of the longest chain
- $O(n+m)$ memory is used
- $\alpha = \frac{n}{m}$ is called load factor
- Operations run in time $O(c+1)$

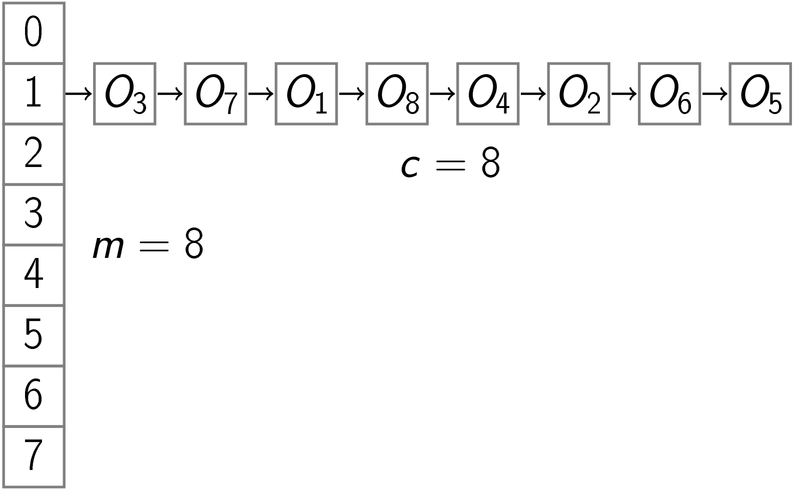
Parameters

- n phone numbers stored
- m — cardinality of the hash function
- c — length of the longest chain
- $O(n + m)$ memory is used
- $\alpha = \frac{n}{m}$ is called load factor
- Operations run in time $O(c + 1)$
- You want small m and c!

Good Example



Bad Example



First Digits

- For the map from phone numbers to names, select $m = 1000$

First Digits

- For the map from phone numbers to names, select $m = 1000$
- Hash function: take first three digits

First Digits

- For the map from phone numbers to names, select $m = 1000$
- Hash function: take first three digits
- $h(800\text{-}123\text{-}45\text{-}67) = 800$

First Digits

- For the map from phone numbers to names, select $m = 1000$
- Hash function: take first three digits
- $h(800\text{-}123\text{-}45\text{-}67) = 800$
- Problem: area code

First Digits

- For the map from phone numbers to names, select $m = 1000$
- Hash function: take first three digits
- $h(800\text{-}123\text{-}45\text{-}67) = 800$
- Problem: area code
- $h(425\text{-}234\text{-}55\text{-}67) =$
  $h(425\text{-}123\text{-}45\text{-}67) =$
  $h(425\text{-}223\text{-}23\text{-}23) = \cdots = 425$

Last Digits

- Select $m = 1000$

Last Digits

- Select $m = 1000$
- Hash function: take last three digits

Last Digits

- Select $m = 1000$
- Hash function: take last three digits
- $h(800\text{-}123\text{-}45\text{-}67) = 567$

Last Digits

- Select $m = 1000$
- Hash function: take last three digits
- $h(800\text{-}123\text{-}45\text{-}67) = 567$
- Problem if many phone numbers end with three zeros

Random Value

- Select $m = 1000$

Random Value

- Select $m = 1000$
- Hash function: random number between 0 and 999

Random Value

- Select $m = 1000$
- Hash function: random number between 0 and 999
- Uniform distribution of hash values

Random Value

- Select $m = 1000$
- Hash function: random number between 0 and 999
- Uniform distribution of hash values
- Different value when hash function called again — we won't be able to find anything!

Random Value

- Select $m = 1000$
- Hash function: random number between 0 and 999
- Uniform distribution of hash values
- Different value when hash function called again — we won't be able to find anything!
- Hash function must be deterministic

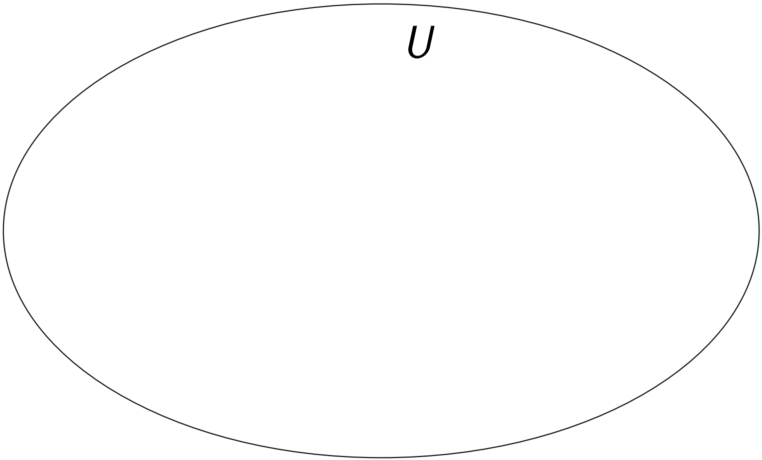
Good Hash Functions

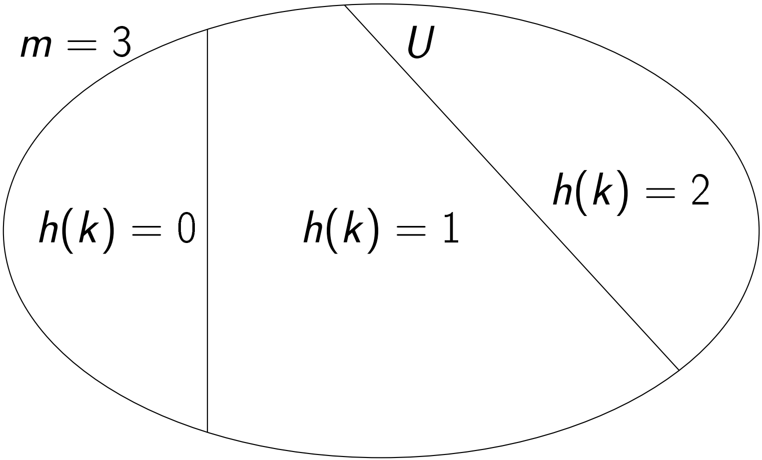
- Deterministic
- Fast to compute
- Distributes keys well into different cells
- Few collisions

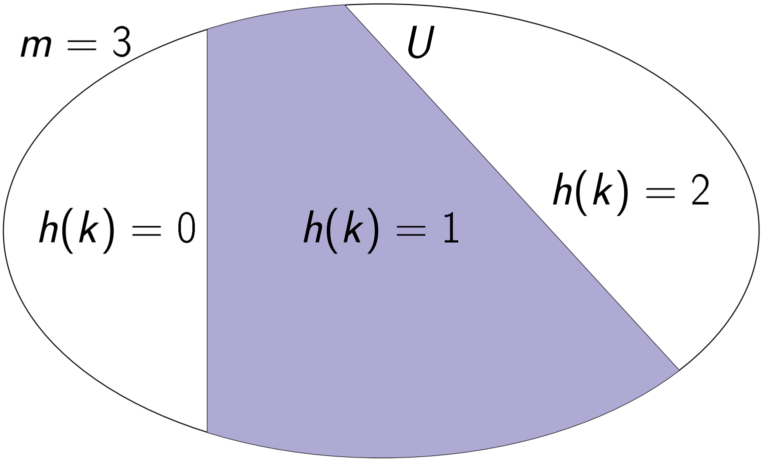
No Universal Hash Function

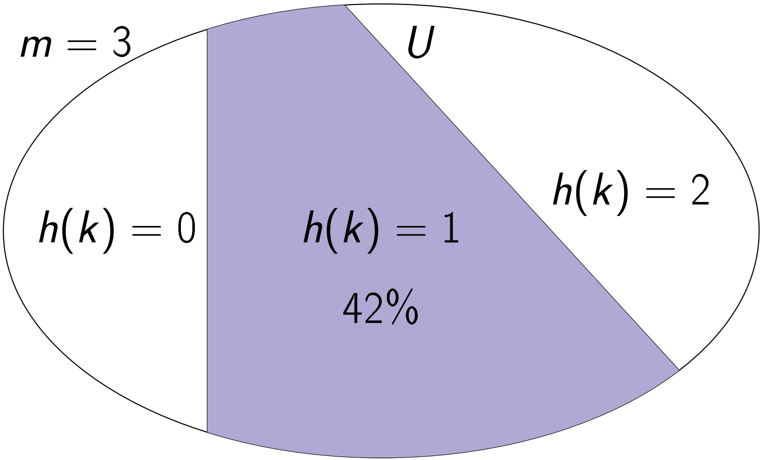
Lemma

If number of possible keys is big ($|U| \gg m$), for any hash function h there is a bad input resulting in many collisions.


U


$m = 3$
U
$h(k) = 0$
$h(k) = 1$
$h(k) = 2$


$m = 3$
U
$h(k) = 0$
$h(k) = 1$
$h(k) = 2$


$m = 3$
U
$h(k) = 0$
$h(k) = 1$
42%
$h(k) = 2$

Outline

1. Good Hash Functions
2. Universal Family
3. Hashing Integers
4. Hashing Strings

Idea

- Remember QuickSort?

Idea

- Remember QuickSort?
- Choosing random pivot helped

Idea

- Remember QuickSort?
- Choosing random pivot helped
- Use randomization!

Idea

- Remember QuickSort?
- Choosing random pivot helped
- Use randomization!
- Define a family (set) of hash functions

Idea

- Remember QuickSort?
- Choosing random pivot helped
- Use randomization!
- Define a family (set) of hash functions
- Choose random function from the family

Universal Family

Definition

Let U be the universe — the set of all possible keys.

Universal Family

Definition

Let U be the universe — the set of all possible keys. A set of hash functions

$$\mathcal{H} = \{h : U \to \{0, 1, 2, \ldots, m - 1\}\}$$

Universal Family

Definition

Let U be the universe — the set of all possible keys. A set of hash functions

$$\mathcal{H} = \{h : U \to \{0, 1, 2, \ldots, m-1\}\}$$

is called a universal family if

Universal Family

Definition

Let U be the universe — the set of all possible keys. A set of hash functions

$$\mathcal{H} = \{h : U \to \{0, 1, 2, \ldots, m-1\}\}$$

is called a universal family if for any two keys $x, y \in U, x \neq y$ the probability of collision

$$Pr[h(x) = h(y)] \leq \frac{1}{m}$$

Universal Family

$$Pr[h(x) = h(y)] \leq \frac{1}{m}$$

means that a collision $h(x) = h(y)$ on selected keys x and y, $x \neq y$ happens for no more than $\frac{1}{m}$ of all hash functions $h \in \mathcal{H}$.

How Randomization Works

- $h(x) = \mathtt{random}(\{0, 1, 2, \ldots, m-1\})$ gives probability of collision exactly $\frac{1}{m}$.

How Randomization Works

- $h(x) = \mathtt{random}(\{0, 1, 2, \ldots, m-1\})$ gives probability of collision exactly $\frac{1}{m}$.
- It is not deterministic — can't use it.

How Randomization Works

- $h(x) = \texttt{random}(\{0, 1, 2, \ldots, m-1\})$ gives probability of collision exactly $\frac{1}{m}$.
- It is not deterministic — can't use it.
- All hash functions in $\mathcal{H}$ are deterministic

How Randomization Works

- $h(x) = \texttt{random}(\{0, 1, 2, \ldots, m-1\})$ gives probability of collision exactly $\frac{1}{m}$.
- It is not deterministic — can't use it.
- All hash functions in $\mathcal{H}$ are deterministic
- Select a random function h from $\mathcal{H}$

How Randomization Works

- $h(x) = \texttt{random}(\{0, 1, 2, \ldots, m-1\})$ gives probability of collision exactly $\frac{1}{m}$.
- It is not deterministic — can't use it.
- All hash functions in $\mathcal{H}$ are deterministic
- Select a random function h from $\mathcal{H}$
- Fixed h is used throughout the algorithm

Running Time

Lemma

If h is chosen randomly from a universal family, the average length of the longest chain c is $O(1+\alpha)$, where $\alpha = \frac{n}{m}$ is the load factor of the hash table.

Corollary

If h is from universal family, operations with hash table run on average in time $O(1+\alpha)$.

Choosing Hash Table Size

- Control amount of memory used with m

Choosing Hash Table Size

- Control amount of memory used with m
- Ideally, load factor $0.5 < \alpha < 1$

Choosing Hash Table Size

- Control amount of memory used with m
- Ideally, load factor $0.5 < \alpha < 1$
- Use $O(m) = O(\frac{n}{\alpha}) = O(n)$ memory to store n keys

Choosing Hash Table Size

- Control amount of memory used with m
- Ideally, load factor $0.5 < \alpha < 1$
- Use $O(m) = O(\frac{n}{\alpha}) = O(n)$ memory to store n keys
- Operations run in time $O(1 + \alpha) = O(1)$ on average

Dynamic Hash Tables

- What if number of keys n is unknown in advance?

Dynamic Hash Tables

- What if number of keys n is unknown in advance?
- Start with very big hash table?

Dynamic Hash Tables

- What if number of keys n is unknown in advance?
- Start with very big hash table?
- You will waste a lot of memory

Dynamic Hash Tables

- What if number of keys n is unknown in advance?
- Start with very big hash table?
- You will waste a lot of memory
- Copy the idea of dynamic arrays!

Dynamic Hash Tables

- What if number of keys n is unknown in advance?
- Start with very big hash table?
- You will waste a lot of memory
- Copy the idea of dynamic arrays!
- Resize the hash table when α becomes too large

Dynamic Hash Tables

- What if number of keys n is unknown in advance?
- Start with very big hash table?
- You will waste a lot of memory
- Copy the idea of dynamic arrays!
- Resize the hash table when α becomes too large
- Choose new hash function and rehash all the objects

Keep load factor below 0.9:

Rehash(T)

$loadFactor \leftarrow \frac{T.\texttt{numberOfKeys}}{T.\texttt{size}}$

if $loadFactor > 0.9$:

  Create T_{new} of size $2 \times T.\texttt{size}$

  Choose h_{new} with cardinality $T_{new}.\texttt{size}$

  For each object O in T:

    Insert O in T_{new} using h_{new}

  $T \leftarrow T_{new}, h \leftarrow h_{new}$

Rehash Running Time

You should call `Rehash` after each operation with the hash table

Similarly to dynamic arrays, single rehashing takes $O(n)$ time, but amortized running time of each operation with hash table is still $O(1)$ on average, because rehashing will be rare

Outline

1. Good Hash Functions
2. Universal Family
3. Hashing Integers
4. Hashing Strings

- Take phone numbers up to length 7, for example 148-25-67

- Take phone numbers up to length 7, for example 148-25-67
- Convert phone numbers to integers from 0 to $10^7 - 1 = 9\,999\,999$:
  $148\text{-}25\text{-}67 \to 1\,482\,567$

- Take phone numbers up to length 7, for example 148-25-67
- Convert phone numbers to integers from 0 to $10^7 - 1 = 9\,999\,999$: $148\text{–}25\text{–}67 \to 1\,482\,567$
- Choose prime number bigger than 10^7, e.g. $p = 10\,000\,019$

- Take phone numbers up to length 7, for example 148-25-67
- Convert phone numbers to integers from 0 to $10^7 - 1 = 9\,999\,999$:
  $148\text{–}25\text{–}67 \to 1\,482\,567$
- Choose prime number bigger than 10^7, e.g. $p = 10\,000\,019$
- Choose hash table size, e.g. $m = 1\,000$

Hashing Integers

Lemma

$\mathcal{H}_p = \left\{ h_p^{a,b}(x) = ((ax + b) \bmod p) \bmod m \right\}$
for all $a, b : 1 \leq a \leq p - 1, 0 \leq b \leq p - 1$
is a universal family

Hashing Phone Numbers

Example

Select $a = 34$, $b = 2$, so $h = h_p^{34,2}$ and consider $x = 1\,482\,567$ corresponding to phone number 148-25-67. $p = 10\,000\,019$.

Hashing Phone Numbers

Example

Select $a = 34$, $b = 2$, so $h = h_p^{34,2}$ and consider $x = 1\ 482\ 567$ corresponding to phone number 148-25-67. $p = 10\ 000\ 019$.

$$(34 \times 1482567 + 2) \bmod 10000019 = 407185$$

Hashing Phone Numbers

Example

Select $a = 34$, $b = 2$, so $h = h_p^{34,2}$ and consider $x = 1\ 482\ 567$ corresponding to phone number 148-25-67. $p = 10\ 000\ 019$.

$(34 \times 1482567 + 2) \bmod 10000019 = 407185$

$407185 \bmod 1000 = 185$

Hashing Phone Numbers

Example

Select $a = 34$, $b = 2$, so $h = h_p^{34,2}$ and consider $x = 1\ 482\ 567$ corresponding to phone number 148-25-67. $p = 10\ 000\ 019$.

$(34 \times 1482567 + 2) \bmod 10000019 = 407185$

$407185 \bmod 1000 = 185$

$h(x) = 185$

General Case

- Define maximum length L of a phone number

General Case

- Define maximum length L of a phone number
- Convert phone numbers to integers from 0 to $10^L - 1$

General Case

- Define maximum length L of a phone number
- Convert phone numbers to integers from 0 to $10^L - 1$
- Choose prime number $p > 10^L$

General Case

- Define maximum length L of a phone number
- Convert phone numbers to integers from 0 to $10^L - 1$
- Choose prime number $p > 10^L$
- Choose hash table size m

General Case

- Define maximum length L of a phone number
- Convert phone numbers to integers from 0 to $10^L - 1$
- Choose prime number $p > 10^L$
- Choose hash table size m
- Choose random hash function from universal family $\mathcal{H}_p$ (choose random $a \in [1, p-1]$ and $b \in [0, p-1]$)

Outline

1. Good Hash Functions
2. Universal Family
3. Hashing Integers
4. Hashing Strings

Lookup Phone Numbers by Name

- Now we need to implement the Map from names to phone numbers

Lookup Phone Numbers by Name

- Now we need to implement the Map from names to phone numbers
- Can also use chaining

Lookup Phone Numbers by Name

- Now we need to implement the Map from names to phone numbers
- Can also use chaining
- Need a hash function defined on names

Lookup Phone Numbers by Name

- Now we need to implement the Map from names to phone numbers
- Can also use chaining
- Need a hash function defined on names
- Hash arbitrary strings of characters

Lookup Phone Numbers by Name

- Now we need to implement the Map from names to phone numbers
- Can also use chaining
- Need a hash function defined on names
- Hash arbitrary strings of characters
- You will learn how string hashing is implemented in Java!

String Length Notation

Definition

Denote by $|S|$ the length of string S.

Examples

$|\texttt{``a''}| = 1$

$|\texttt{``ab''}| = 2$

$|\texttt{``abcde''}| = 5$

Hashing Strings

- Given a string S, compute its hash value

Hashing Strings

- Given a string S, compute its hash value
- $S = S[0]S[1]\dots S[|S|-1]$, where $S[i]$ — individual characters

Hashing Strings

- Given a string S, compute its hash value
- $S = S[0]S[1]\dots S[|S|-1]$, where $S[i]$ — individual characters
- We should use all the characters in the hash function

Hashing Strings

- Given a string S, compute its hash value
- $S = S[0]S[1]\ldots S[|S|-1]$, where $S[i]$ — individual characters
- We should use all the characters in the hash function
- Otherwise there will be many collisions:

Hashing Strings

- Given a string S, compute its hash value
- $S = S[0]S[1]\dots S[|S|-1]$, where $S[i]$ — individual characters
- We should use all the characters in the hash function
- Otherwise there will be many collisions:
- For example, if $S[0]$ is not used, $h(\text{``aa''}) = h(\text{``ba''}) = \dots = h(\text{``za''})$

Preparation

- Convert each character $S[i]$ to integer code

Preparation

- Convert each character $S[i]$ to integer code
- ASCII code, Unicode, etc.

Preparation

- Convert each character $S[i]$ to integer code
- ASCII code, Unicode, etc.
- Choose big prime number p

Polynomial Hashing

Definition

Family of hash functions

$$\mathcal{P}_p = \left\{ h_p^x(S) = \sum_{i=0}^{|S|-1} S[i]x^i \bmod p \right\}$$

with a fixed prime p and all $1 \leq x \leq p-1$ is called polynomial.

`PolyHash`(S, p, x)

```
hash ← 0
for i from |S| − 1 down to 0:
    hash ← (hash × x + S[i]) mod p
return hash
```

Example: $|S| = 3$

1. $\texttt{hash} = 0$
2. $\texttt{hash} = S[2] \bmod p$
3. $\texttt{hash} = S[1] + S[2]x \bmod p$
4. $\texttt{hash} = S[0] + S[1]x + S[2]x^2 \bmod p$

Java Implementation

The method `hashCode` of the built-in Java class `String` is very similar to our `PolyHash`, it just uses $x = 31$ and for technical reasons avoids the $\pmod p$ operator.

Java Implementation

The method `hashCode` of the built-in Java class `String` is very similar to our `PolyHash`, it just uses $x = 31$ and for technical reasons avoids the $\pmod{p}$ operator.

You now know how a function that is used trillions of times a day in many thousands of programs is implemented!

Lemma

For any two different strings s_1 and s_2 of length at most $L+1$, if you choose h from $\mathcal{P}_p$ at random (by selecting a random $x \in [1, p-1]$), the probability of collision $Pr[h(s_1) = h(s_2)]$ is at most $\frac{L}{p}$.

Proof idea

This follows from the fact that the equation $a_0 + a_1 x + a_2 x^2 + \cdots + a_L x^L = 0 \pmod{p}$ for prime p has at most L different solutions x.

Cardinality Fix

For use in a hash table of size m, we need a hash function of cardinality m.

First apply random h from $\mathcal{P}_p$ and then hash the resulting value again using integer hashing. Denote the resulting function by h_m.

Lemma

For any two different strings s_1 and s_2 of length at most $L+1$ and cardinality m, the probability of collision $Pr[h_m(s_1) = h_m(s_2)]$ is at most $\frac{1}{m} + \frac{L}{p}$.

Polynomial Hashing

Corollary

If $p > mL$, for any two different strings s_1 and s_2 of length at most $L+1$ the probability of collision $Pr[h_m(s_1) = h_m(s_2)]$ is $O(\frac{1}{m})$.

Proof

$$\frac{1}{m} + \frac{L}{p} < \frac{1}{m} + \frac{L}{mL} = \frac{1}{m} + \frac{1}{m} = \frac{2}{m} = O\left(\frac{1}{m}\right) \qquad \square$$

Running Time

- For big enough p again have $c = O(1 + \alpha)$

Running Time

- For big enough p again have $c = O(1 + \alpha)$
- Computing `PolyHash`(S) runs in time $O(|S|)$

Running Time

- For big enough p again have $c = O(1 + \alpha)$
- Computing `PolyHash`(S) runs in time $O(|S|)$
- If lengths of the names in the phone book are bounded by constant L, computing $h(S)$ takes $O(L) = O(1)$ time

Conclusion

- You learned how to hash integers and strings
- Phone book can be implemented as two hash tables
- Mapping phone numbers to names and back
- Search and modification run on average in $O(1)$!