Universal Family of Hash Function

261217 Data Structures for Computer Engineers

Patiwet Wuttisarnwattana, Ph.D.

patiwet@eng.cmu.ac.th

Computer Engineering, Chiang Mai University

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.

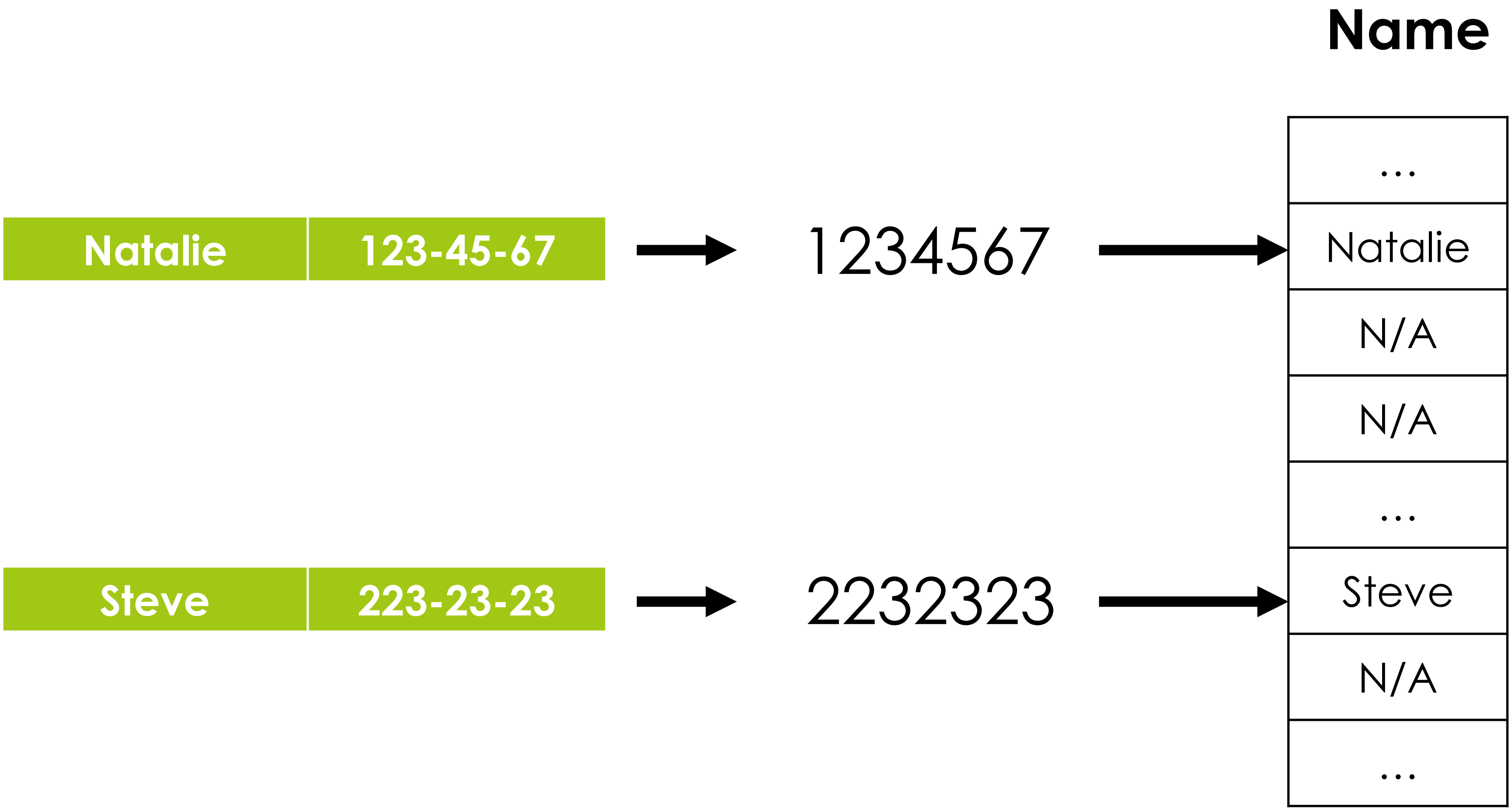
Mapping

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

Direct Addressing

- int(123-45-67) = 1,234,567
- 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***[int(P)]
- If no contact with phone number P, ***Name***[int(P)] = N/A

Direct Addressing



Direct Addressing

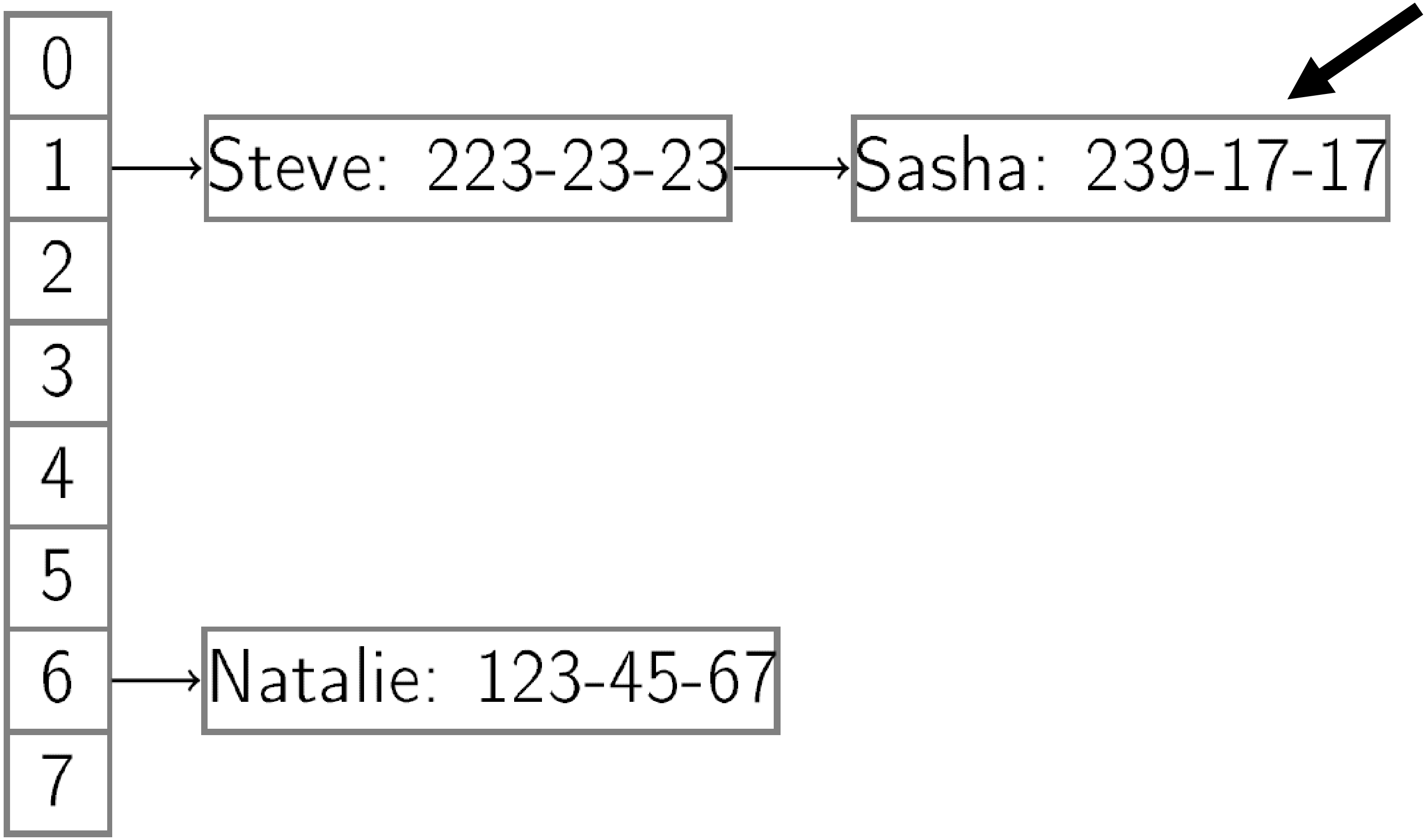
- Operations run in ***O***(1)
- Memory usage: ***O***(10^L), where L is the maximum length of a phone number
- There will no collision (Good)
- Problematic with international numbers of length 12 and more: we will need 10^{12} bytes = 1TB to store one person's phone book – this won't fit in anyone's phone!

Collision Resolution using 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***(int(*P*))] contains the name for phone number *P*
- If there is a collision after the hashing, you just append the new object after the previous one (addBack)

Chaining

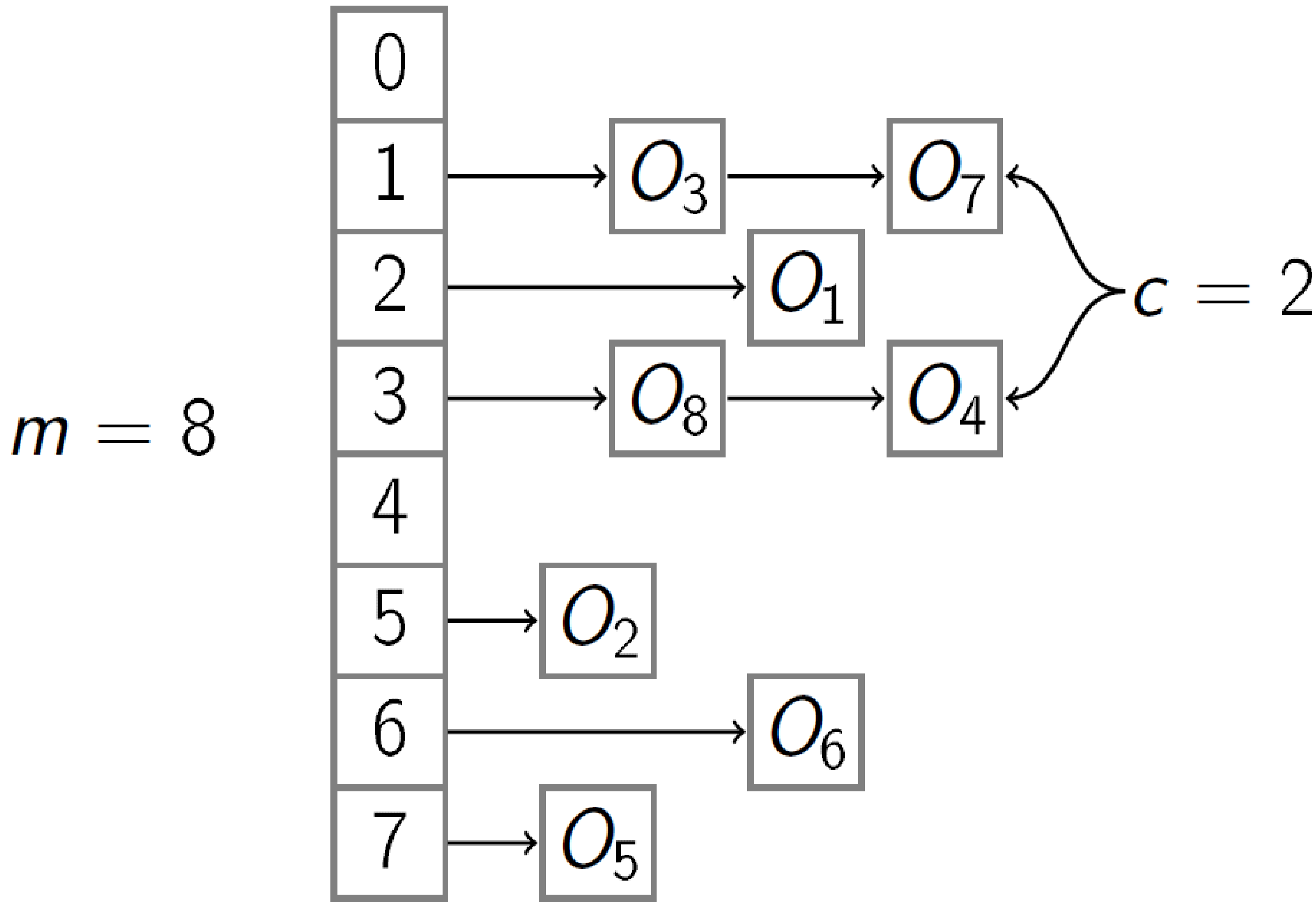
Map to the same index
Collision!!!
AddBack



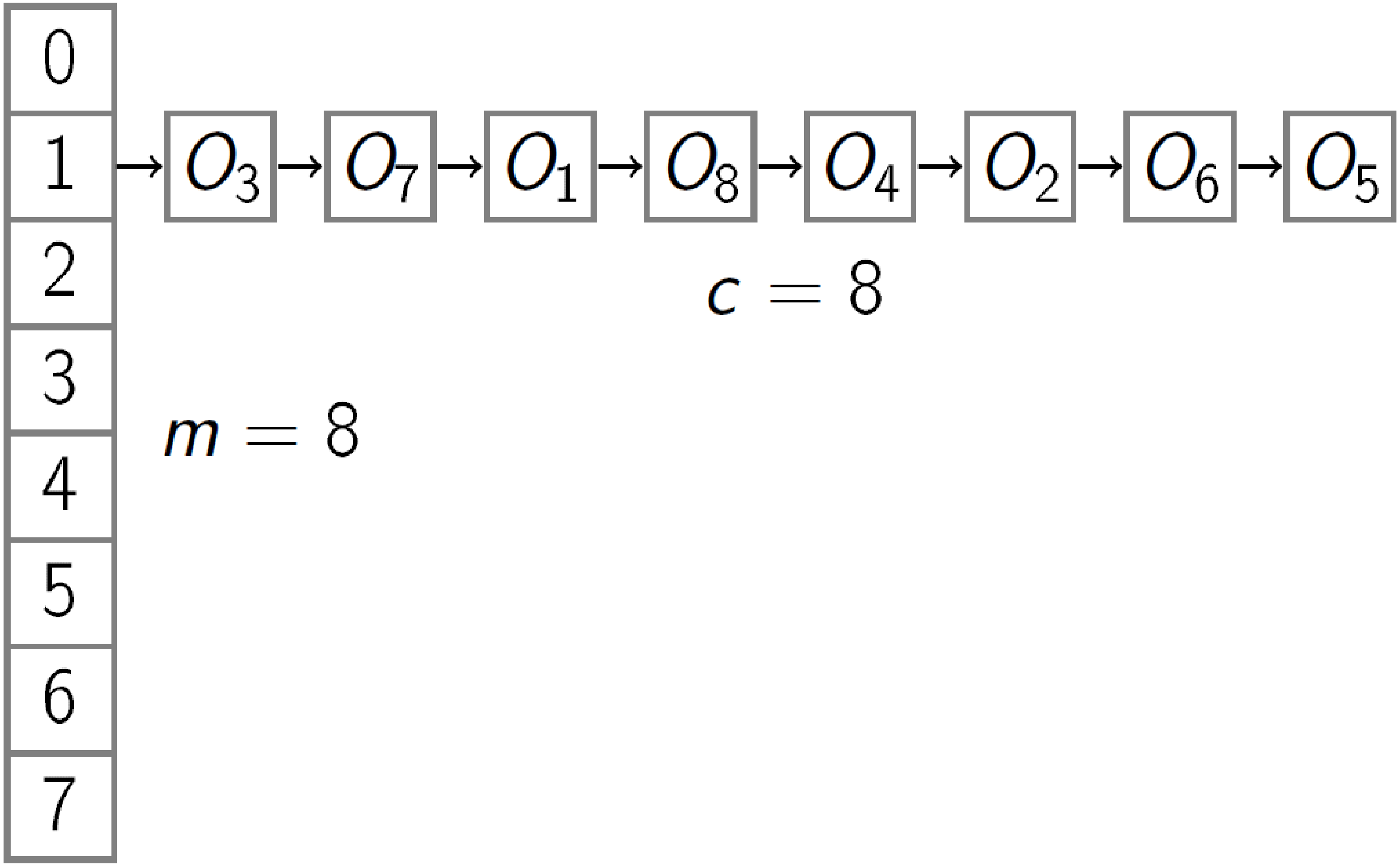
Parameters

- ***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
- Hash function: take first three digits
- h(800-123-45-67) = 800
- Problem: area code
- h(425-234-55-67) =
  h(425-123-45-67) =
  h(425-223-23-23) = … = 425

Last digits

- Select ***m***=1000
- Hash function: take last three digits
- h(800-123-45-67) = 567
- Problem if many phone numbers end with three zeros, 555, 888, 999, …

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

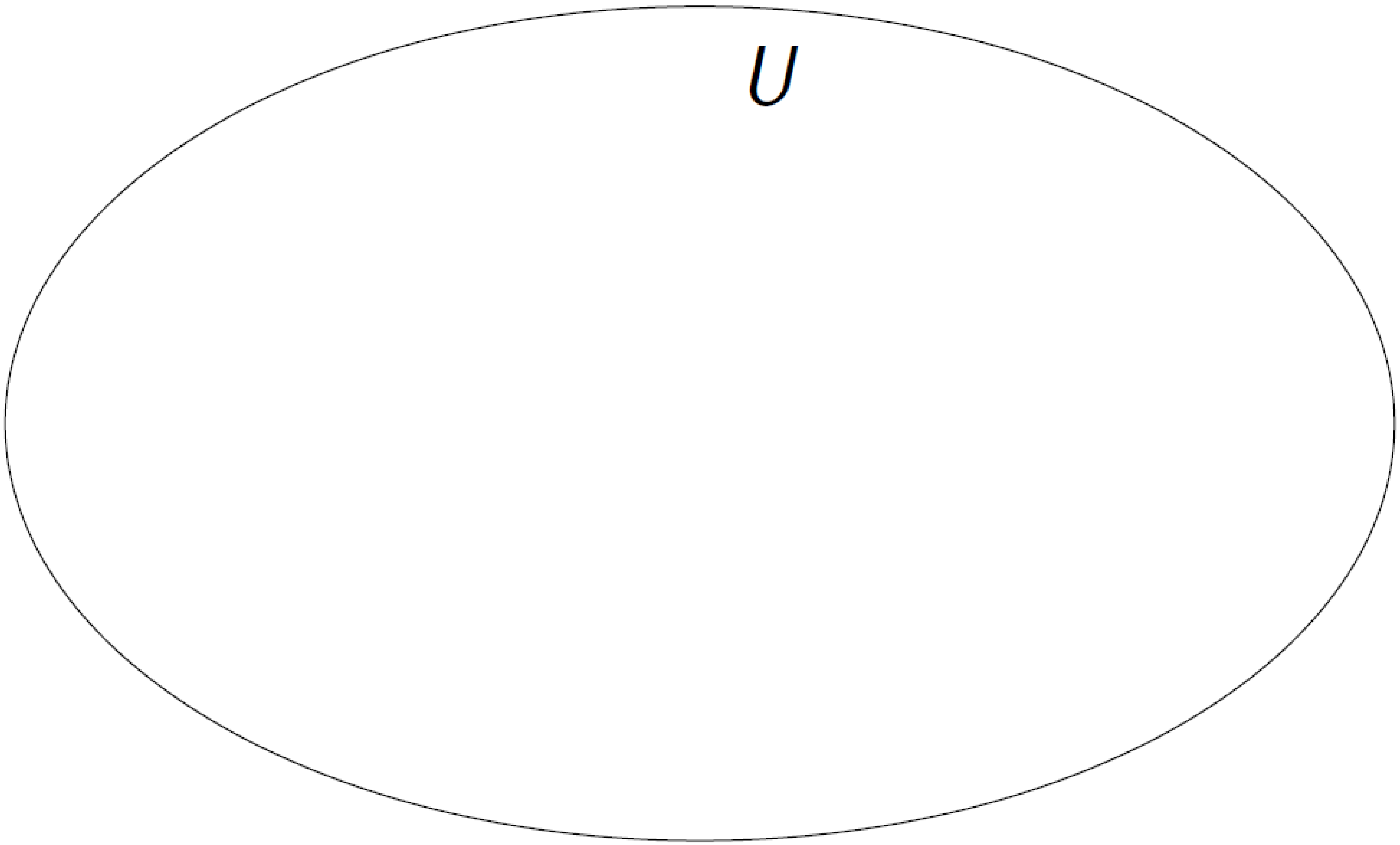
Good Hash Functions

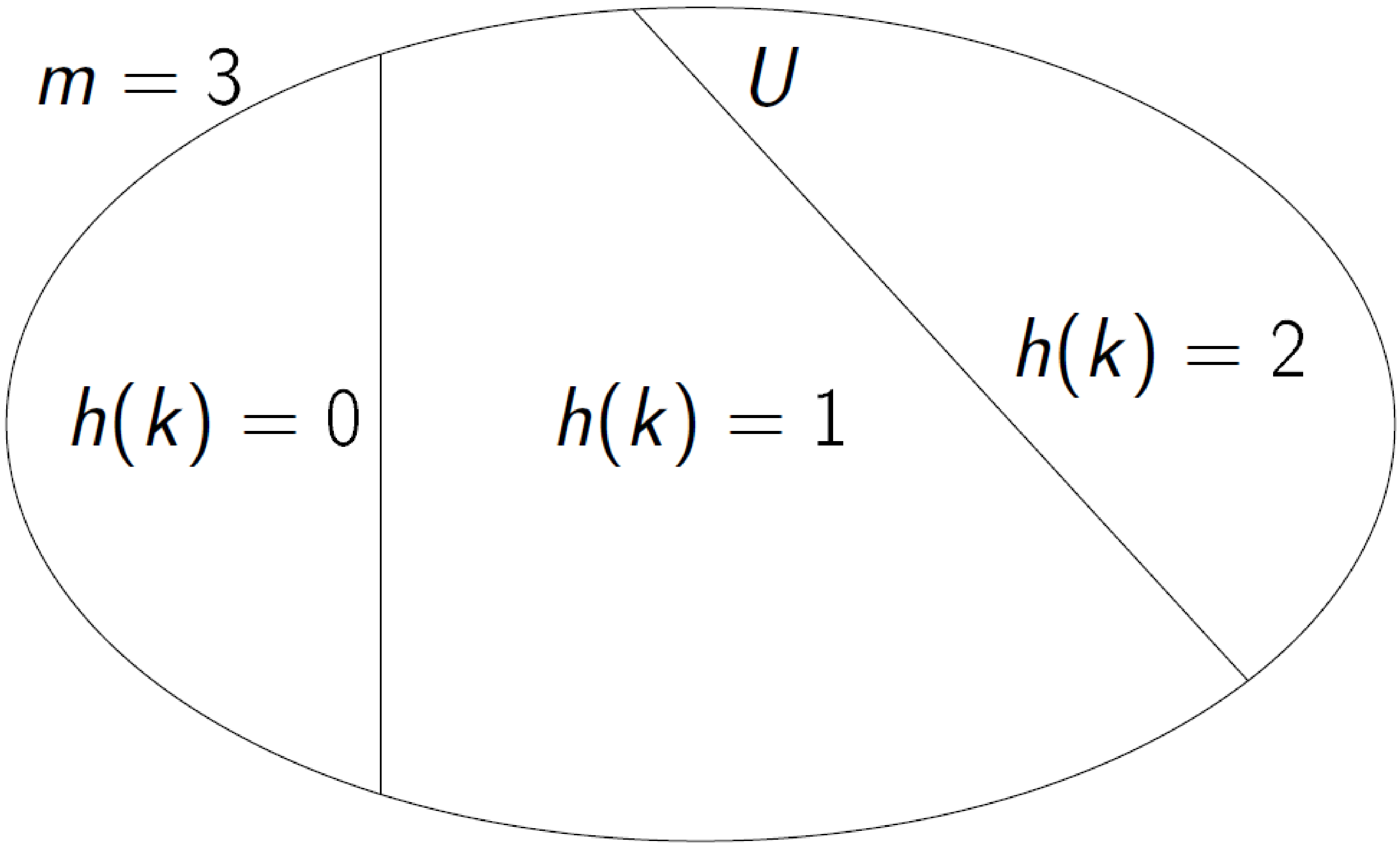
- Deterministic
- Fast to compute
- Distributes keys well into different cells
- Few collisions

There is no best hash function that guarantee no collision

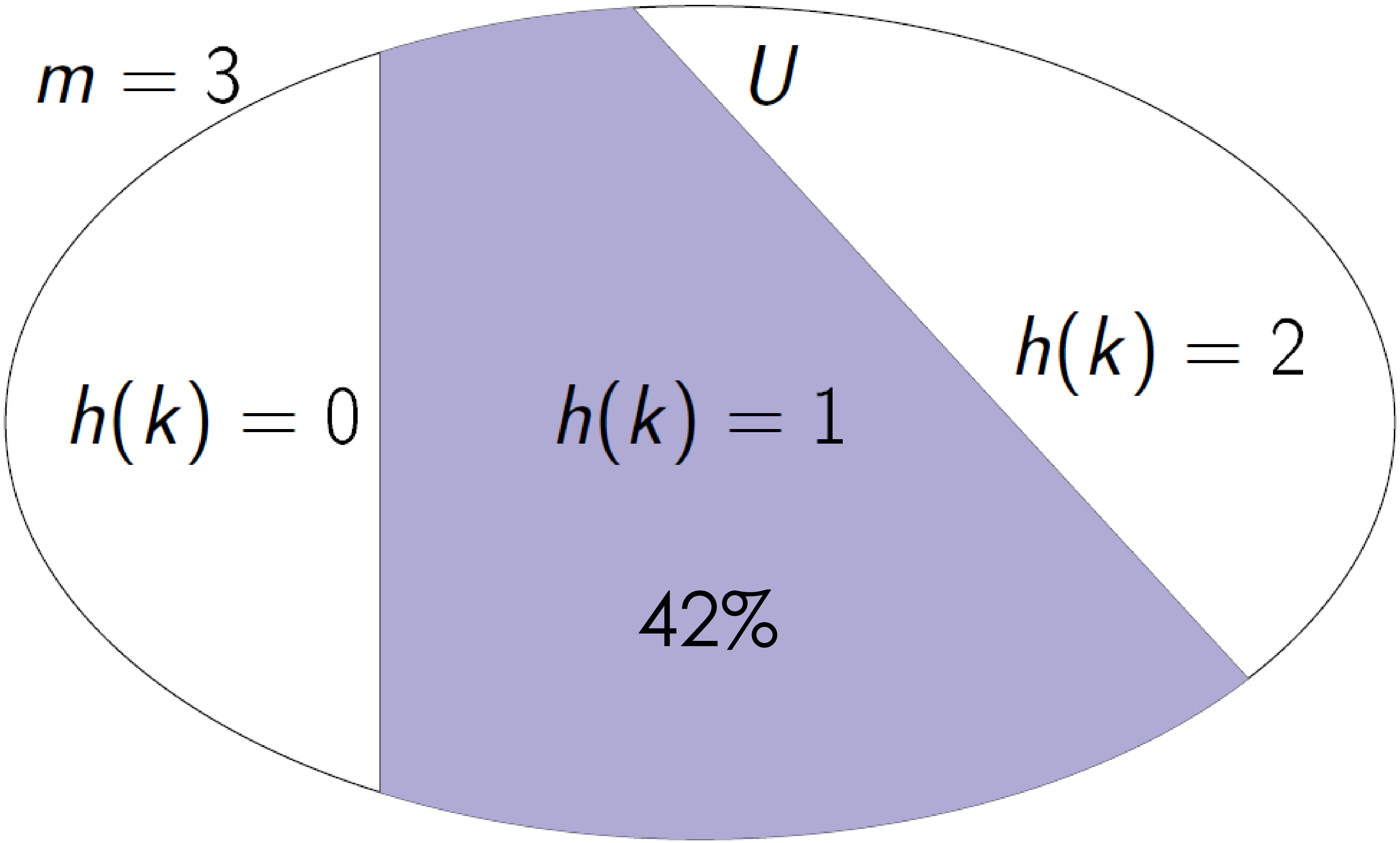
Lemma

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
$h(k) = 2$
$h(k) = 0$
$h(k) = 1$


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

Universal Family of Hash Function

- Definition
- Let U be the universe – the set of all possible keys.
- A set of hash functions

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

- 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

- $h(x) = random(\{0, 1, 2, \dots, m-1\})$ 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
- Ideally, load factor $0.5 < \alpha < 1.0$
- Use $O(m) = O\left(\frac{n}{\alpha}\right) = O(n)$ 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?
- 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 × T.size
  Choose h_new with cardinality T_new.size
  For each object O in T:
    Insert O in T_new using h_new
  T ← T_new, h ← 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

Numerical Object Hashing Algorithm

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

Linear Transformation Hashing

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

p is a prime number greater than |U|
Parameters ***a*** and ***b*** can be chosen randomly between 1|0 and p-1
m is the hash table size

Example: Hashing Phone Number

- Select ***a*** = 34, ***b*** = 2, ***p*** = 10,000,019, so $h \ = \ h_p^{34,2}$
- Consider x = 1,482,567 corresponding to phone number “148-25-67”
- (34 x 1482567 + 2) mod 10000019 = 407185
- 407185 mod 1000 = 185
- $h(x)$ = 185

General Algorithm

- 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 $b \in [0, p-1]$)

String Hashing Algorithm

- We want to 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

String Length Notation

Definition

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

Examples

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

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

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

Hashing Strings

- Given a string $\boldsymbol{S}$, compute its hash value
- $\boldsymbol{S}$ = S[0]S[1]S[2] … S[|$\boldsymbol{S}$|-1], where S[i] are 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("aa") = h("ba") = … = h("za")

Character to number

- Preparation step is to convert each character S[i] to integer
- ASCII code, Unicode, etc.
  - ‘0’ = 48, ‘1’ = 49, ‘2’ = 50
  - ‘A’ = 65, ‘B’ = 66, ‘C’ = 67
  - ‘a’ = 97, ‘b’ = 98, ‘c’ = 99
- You also need to choose a big prime number ***p***

Polynomial Hashing

Polynomial Transformation

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.

Parameter x is randomly chosen between 1 and p - 1

$\texttt{PolyHash}(S, p, x)$

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

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 Dynamic Hash Table

- 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 $O(L) = 0(1)$ 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)