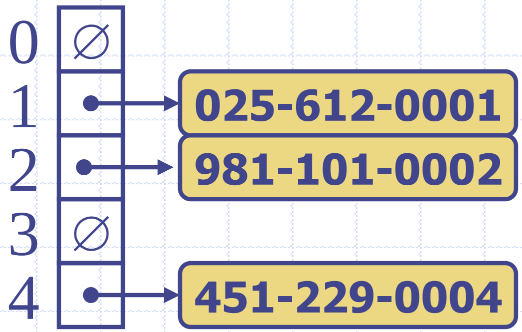
Hash Tables



Maps in STL

- Two types of maps in STL
  - #include <map>
    - Sorted keys
    - Implementation based on trees
    - Complexity in search: O(log(n))
  - #include <unordered_map>
    - Unsorted keys
    - Implementation based on hash tables
    - Complexity in search: O(1)

Aka associative arrays

Hash Functions and Hash Tables

Strings or numbers

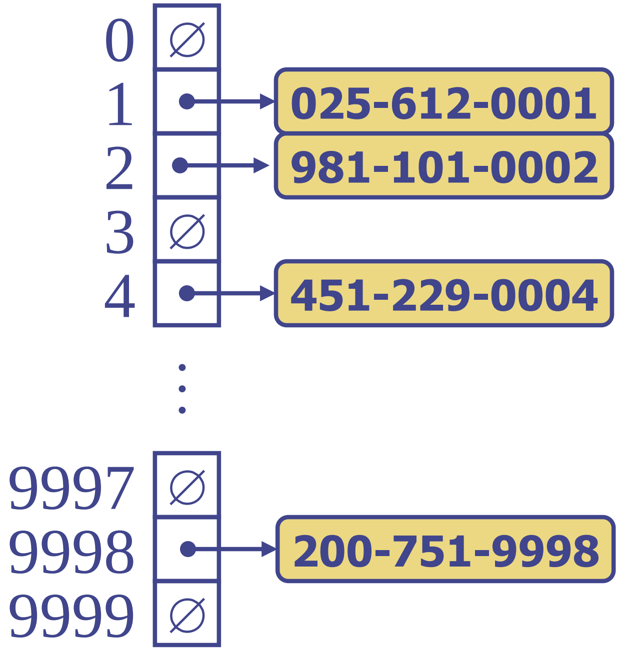
- Hash functions/tables: A common way to implement maps
- A hash function h maps keys to integers in an interval $[0, N - 1]$
  - Example: $h(x) = x \bmod N$ is a hash function for integer keys
  - The integer $h(x)$ is called the hash value of key x
- A hash table for a given key type consists of
  - A hash function
  - An array (aka bucket array, or table) of size N where we can store item (k, o) at index $i = h(k)$

Key

Data

Example

- We design a hash table for a map storing entries as (SSN, Name), where SSN (social security number) is a ten-digit positive integer
- Our hash table uses an array of size $N = 10{,}000$ and the hash function
  $h(x) = \text{last four digits of } x$



Another example:
Dispatch of quiz exam sheets

Hash Functions

Quiz!

- A hash function is usually specified as the composition of two functions:

  Hash code:

  h_1: keys $\rightarrow$ integers

  Words for a dictionary

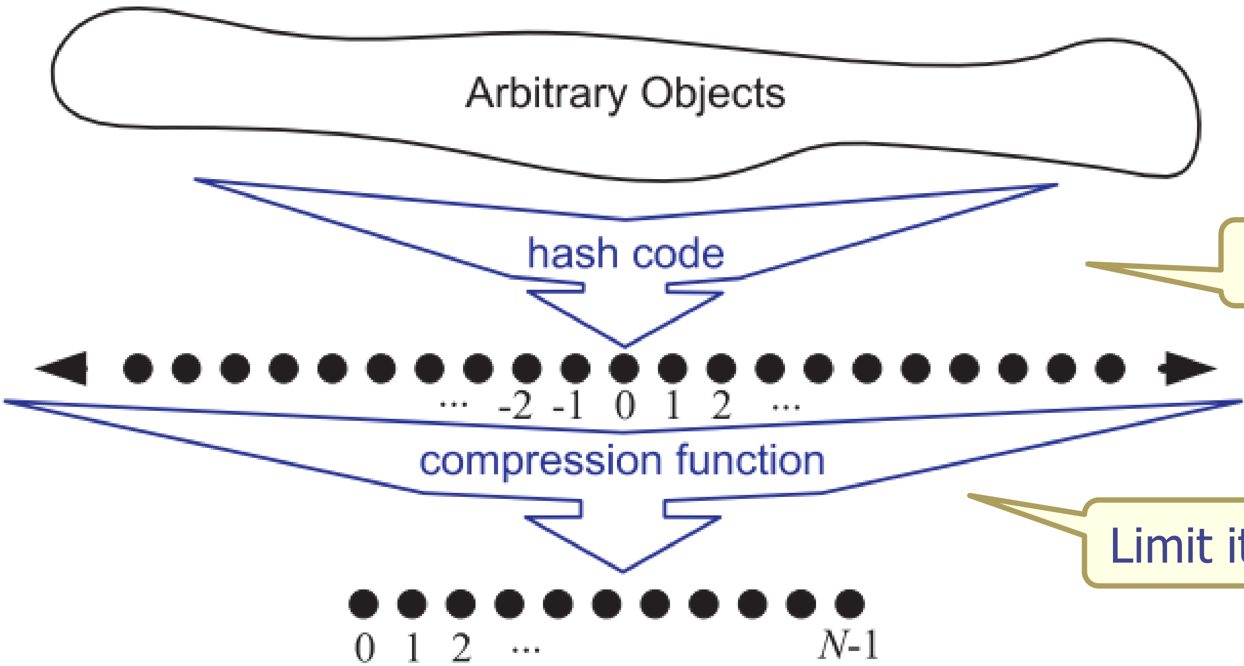
  Compression function:

  h_2: integers $\rightarrow [0, N-1]$

  Size of the hash table

- The hash function is then specified as $h(x) = h_2(h_1(x))$.
- Goals of the hash function
  - To "disperse" the keys as much as possible.
  - To compute the result ASAP

Two Steps in a Hash Function

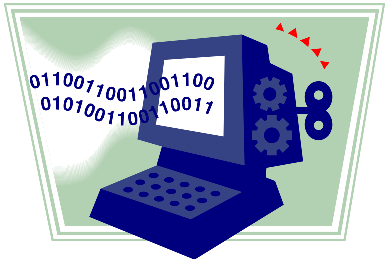


Convert to an integer

Limit its range

In both steps, we need to avoid collision!

Hash Codes



- Integer cast:
  - Reinterpret the bits of the key as an integer
  - Suitable for keys of length less than or equal to the number of bits of the integer type (e.g., byte, short, int and float in C++)
- Component sum:
  - Partition the bits of the key into components of fixed length (e.g., 16 or 32 bits) and sum the components (ignoring overflows)
  - Suitable for numeric keys of fixed length greater than or equal to the number of bits of the integer type (e.g., long and double in C++)

tomorrow
tomorrow

Collisions from simple sum:
stop, tops, pots, spot

Hash Codes (cont.)

- Polynomial hash code
  - Partition the key into a vec. of components of fixed length (e.g., 8, 16 or 32 bits)

    $$a_0\, a_1\, \ldots\, a_{n-1}$$
  - Evaluate the polynomial

    $$p(z) = a_0 + a_1 z + a_2 z^2 + \ldots + a_{n-1} z^{n-1}$$

    at a value z, ignoring overflow
  - Especially suitable for strings (e.g., the choice $z = 33$ gives at most 6 collisions on a set of 50,000 English words)

'csie' ➔ $99+115z+105z^2+101z^3$

- Polynomial $p(z)$ can be evaluated in $O(n)$ time using Horner's rule:

Quiz!

  - Successively computation from the previous one in $O(1)$ time

    $$p_0(z) = a_{n-1}$$

    $$p_i(z) = a_{n-i-1} + z p_{i-1}(z)$$

    $$(i = 1, 2, \ldots, n-1)$$
  - $p(z) = p_{n-1}(z)$

'csie' ➔ 99+(115+(105+101z)z)z

Ignore overflow!

Compression Functions

Most commonly used! (Quiz!)

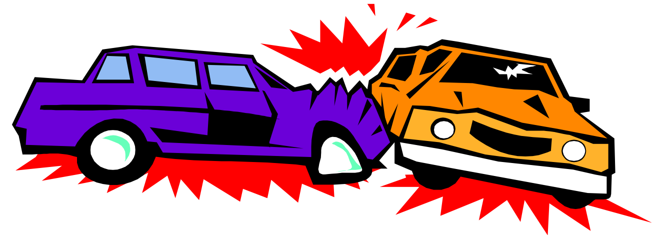
- Division method:
  - $h_2(y) = |y| \bmod N$
  - The size N of the hash table is a prime to avoid collisions
  - The reason has to do with number theory and is beyond the scope of this course

What if y={200, 205, 210, 215, …, 600} and N=100?

- Multiply, Add and Divide (MAD):
  - $h_2(y) = |ay + b| \bmod N$
  - a and b are nonnegative integers such that

    $a \bmod N \neq 0$
  - Otherwise, every integer would map to the same value b

Collision Handling



Quiz!

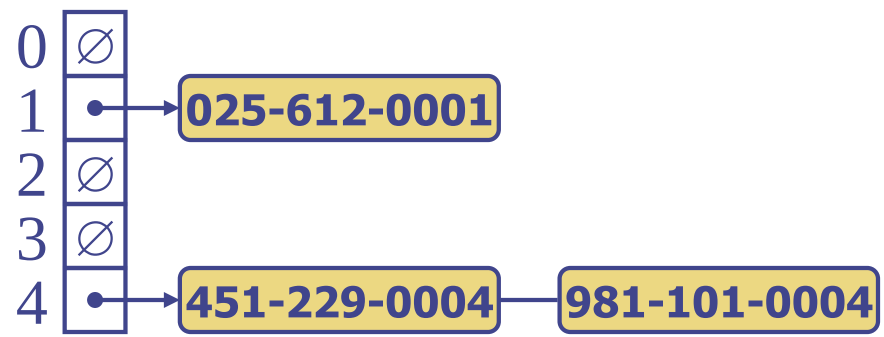
- Definition of collisions

  Different keys mapped to the same index

Quiz!

- Two methods for collision handling
  - Separate chaining
  - Open addressing

- Separate chaining



Each cell points to a linked list
➔ Requires extra memory

Quiz!

Map with Separate Chaining

Delegate operations to a list-based map at each cell:

Algorithm find(k):
return A[h(k)].find(k)

Algorithm put(k,v):
t = A[h(k)].put(k,v)
if t = **null then** {k is a new key}
    n = n + 1
return t

Algorithm erase(k):
t = A[h(k)].erase(k)
if t ≠ **null then** {k was found}
    n = n - 1
return t

Open Addressing

- Open addressing: Place colliding item in another table cell
  - Linear probing: Place in the next (circularly) available cell ➔ Tends to create lump causing a longer sequence of probes
  - Quadratic probing
  - Double hashing
- Example of linear probing:

  Quiz!

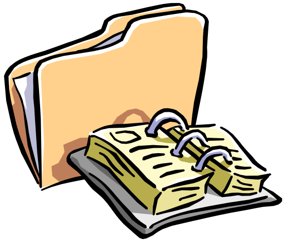
  - $h(k) = k \bmod 13$
  - Insert keys 18, 41, 22, 44, 59, 32, 31, 73

0	1	2	3	4	5	6	7	8	9	10	11	12

⇩

		41			18	44	59	32	22	31	73	
0	1	2	3	4	5	6	7	8	9	10	11	12

Search with Linear Probing



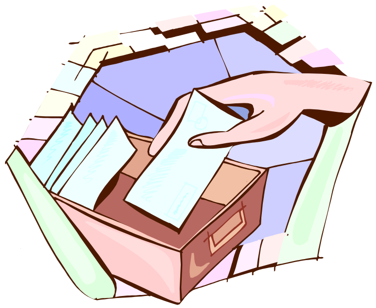
- Consider a hash table A that uses linear probing
- find(k)
  - We start at cell $h(k)$
  - We probe consecutive locations until one of the following occurs
    - An item with key k is found, or
    - An empty cell is found, or
    - N cells have been unsuccessfully probed

Algorithm *find*(k)
  $i \leftarrow h(k)$
  $p \leftarrow 0$
  repeat
    $c \leftarrow A[i]$
    if $c = \varnothing$
      return *null*
    else if $c.key() = k$
      return $c.value()$
    else
      $i \leftarrow (i + 1) \bmod N$
      $p \leftarrow p + 1$
  until $p = N$
  return *null*

Updates with Linear Probing

- To handle insertions and deletions, we introduce a special marker object, called ***AVAILABLE***, which replaces deleted elements
- erase(k)
  - Search for an entry with key k
  - If such an entry (k, o) is found, replace it with the special item ***AVAILABLE*** and return element o
  - Else, return ***null***
- put(k, o)
  - Throw an exception if the table is full
  - Start at cell $h(k)$
  - Probe consecutive cells until one of the following occurs
    - A cell i is found that is either empty or stores ***AVAILABLE*** → put(k, o)
    - N cells have been unsuccessfully probed → Throw an exception

Double Hashing



- Use a secondary hash function $d(k)$ and handles collisions by placing an item in the first available cell of the series

$$(i + j*d(k)) \bmod N$$

for $j = 0, \; 1, \; \ldots \;, N - 1$

i = h(k) = original hash value

- The secondary hash function $d(k)$ cannot have zero values
- Common choice of the secondary hash function:

  $d(k) = q - k \bmod q$, where

  - q is a prime and $q < N$
  - The possible values for $d(k)$ are $1, 2, \ldots, q$

Example of Double Hashing

- Consider a hash table storing integer keys that handles collision with double hashing
  - $N = 13$
  - $h(k) = k \bmod 13$
  - $d(k) = 7 - k \bmod 7$
- Insert keys 18, 41, 22, 44, 59, 32, 31, 73

k	$h(k)$	$d(k)$	Probes		
18	5	3	5		
41	2	1	2		
22	9	6	9		
44	5	5	5	10	
59	7	4	7		
32	6	3	6		
31	5	4	5	9	0
73	8	4	8		

(h(k)+1*d(k)) mod 13

(h(k)+1*d(k)) mod 13

(h(k)+2*d(k)) mod 13

0	1	2	3	4	5	6	7	8	9	10	11	12

⇩

31		41			18	32	59	73	22	44		
0	1	2	3	4	5	6	7	8	9	10	11	12

Quiz!

Comparison of Collision Handling

- Separate chaining
- Open addressing
  - Linear probing

    A[(i+j) mod N], j=0, 1, 2...
  - Quadratic probing

    A[(i+j^2) mod N], j=0, 1, 2...
  - Double hashing

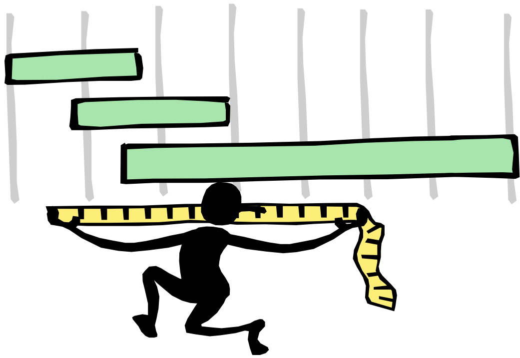
    A[(i+j*d(k)) mod N], j=0, 1, 2...

Most commonly used if memory is big

Quiz!

i = h(k) = original hash value

Performance of Hashing



- In the worst case, searches, insertions and removals on a hash table take $O(n)$ time, which occurs when all the keys inserted into the map collide
- The load factor $\alpha = n/N$ affects the performance of a hash table

Quiz!

- Assuming that the hash values are like random numbers, it can be shown that the expected number of probes for an insertion with open addressing is $1/(1 - \alpha)$

Reference

- The expected running time of all the dictionary ADT operations in a hash table is $O(1)$
- In practice, hashing is very fast provided the load factor is not close to 100%
- Applications of hash tables:
  - small databases
  - browser caches

Summary

- Hash table
  - Hash function
    - Hash code
    - Compression function
  - Bucket array
- Applications
  - Associative arrays for dictionaries, etc.
  - Cache results during DFS or DP, etc.
- Collision handling
  - Separate chaining
  - Open addressing
    - Linear probing
    - Quadratic probing
    - Double hashing

Exercise

Quiz!

- Example of probing:
  - $h(k) = k \bmod 13$
  - Insert keys 16, 41, 29, 45, 60, 28, 55, 23, 32

Linear probing:

0	1	2	3	4	5	6	7	8	9	10	11	12

0	1	2	3	4	5	6	7	8	9	10	11	12
		41	16	29	28	45	55	60	32	23		

Quadratic probing:

0	1	2	3	4	5	6	7	8	9	10	11	12

0	1	2	3	4	5	6	7	8	9	10	11	12
	32	41	16	29		45	55	60		23	28	