THE UNIVERSITY OF TEXAS AT DALLAS

CS3345 Data Structures & Algorithm Analysis

Lecture Notes 15

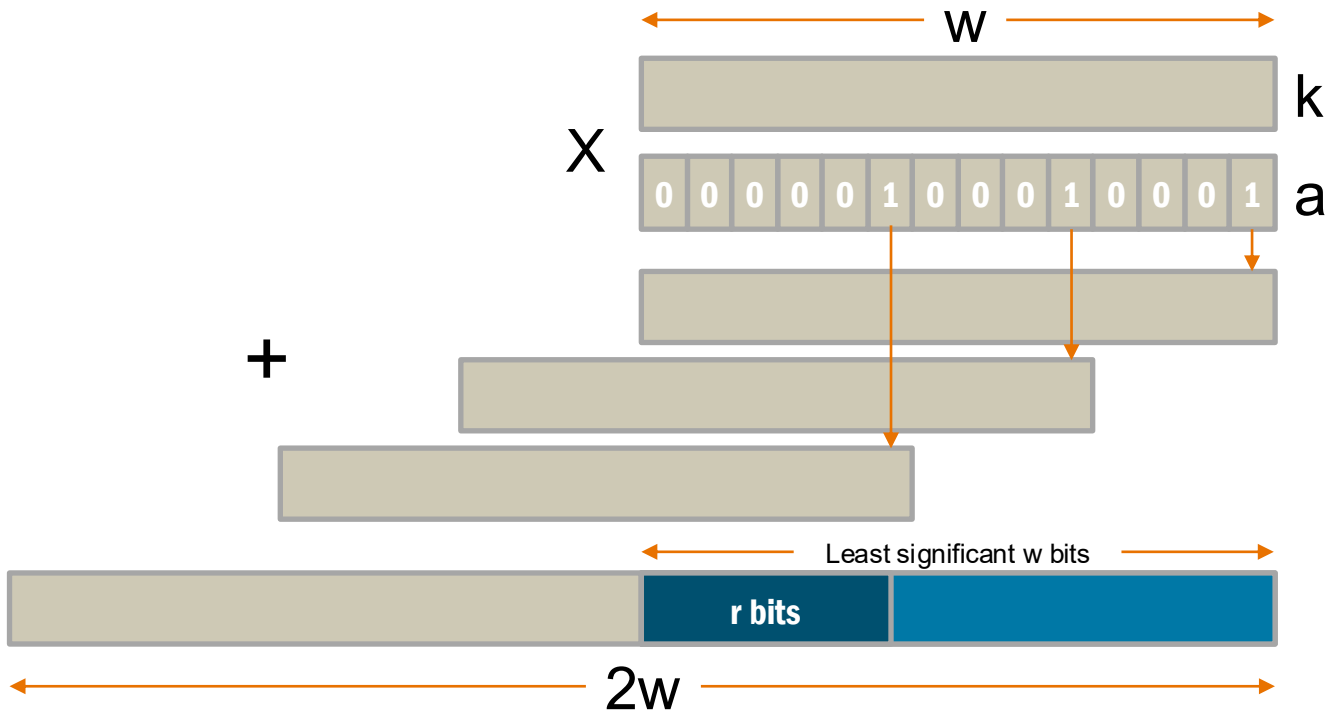
Omar Hamdy
Associate Professor
Department of Computer Science

Review: Hash Map

- Hashing maps data keys to array indexes for a fast O(1) access time.
- This is possible after solving two key problems:
  - Problem 1: Keys are not necessarily non-negative integers all the time. Solved through pre-hashing
  - Problem 2: The inefficient memory use. Solved using hashing
- Pre-hashing: Maps keys to a non-negative integers through different approaches and algorithms
- Hashing: Uses a hash function to transform a key into a table address. A good hashing function:
  - Is easy to compute
  - Approximates a random function: for every input, every output is equally likely (simple uniform hashing)

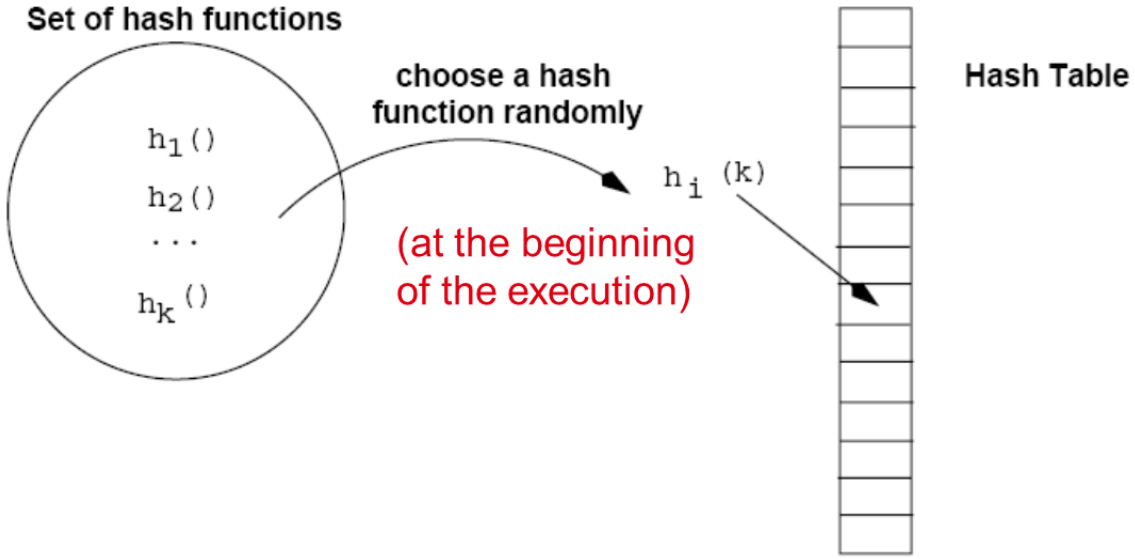
Hashing Function - Multiplication

- There are multiple implementations of the multiplication method
- $h(k) = [(a.k) mod\ 2^w] \gg w - r$
  - a is an odd constant, w is the number of bits that represent integer k, and m is 2^r
  - This means r-bits will produce a number from 0 to m-1



Hashing Function - Universal

- The fact is keys are not randomly distributed. Therefore, the goal of the universal hashing is to produce random table indexes irrespective of the keys.
- Idea is to select a hash function at random, from a designed class of functions at the beginning of the execution



Hashing Function - Universal

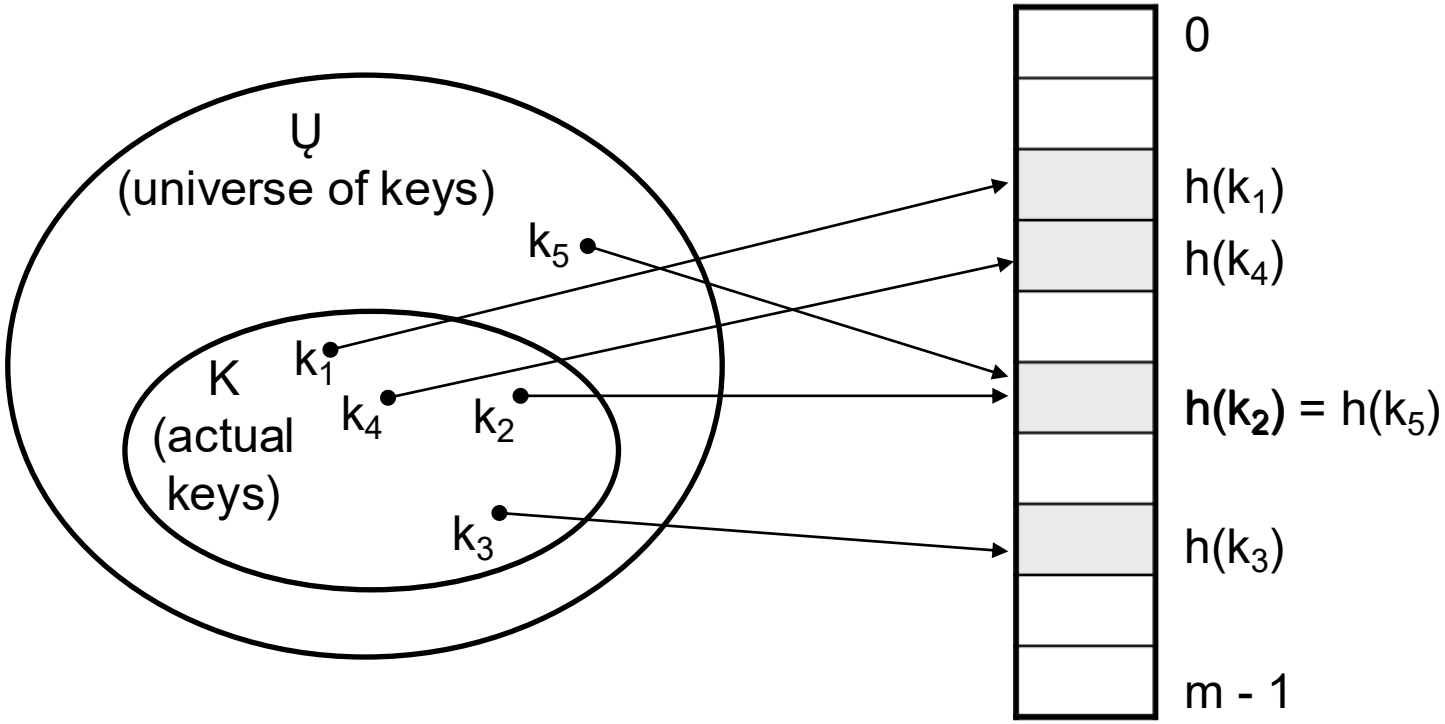
- The random hash function is defined as follows:
- $h(k) = [(ak + b)\ mod\ p]\ mod\ m$
  - a and b are random numbers between {0, …. p-1}
  - p is a prime number > |Ų|
  - The mod m at the end is to ensure you produce a number between 0 and m-1
- It could be proven that for $k_1 \neq k_2$, then at worst-case keys, the probability over the random number $Pr_{a,b}\{h(k_1) = h(k_2)\} = 1/m$

Example – Universal Hashing

- Assume m = 6 and p is chosen to be 17. a and b are randomly chosen to be 3 and 4 respectively.
- $h_{a,b}(k) = [(ak + b)\ mod\ p]\ mod\ m$
- $h_{a,b}(8) = [(3x8 + 4)\ mod\ 17]\ mod\ 6$
- $h_{a,b}(8) = (28\ mod\ 17)\ mod\ 6 = 5$

Hashing Function Problem

- As indicated before, there is a key problem in hashing
  - Pigeonhole concept
  - Collision
  - Collision occurs when the hash values of two distinct keys map to the same index in the array.



Collision

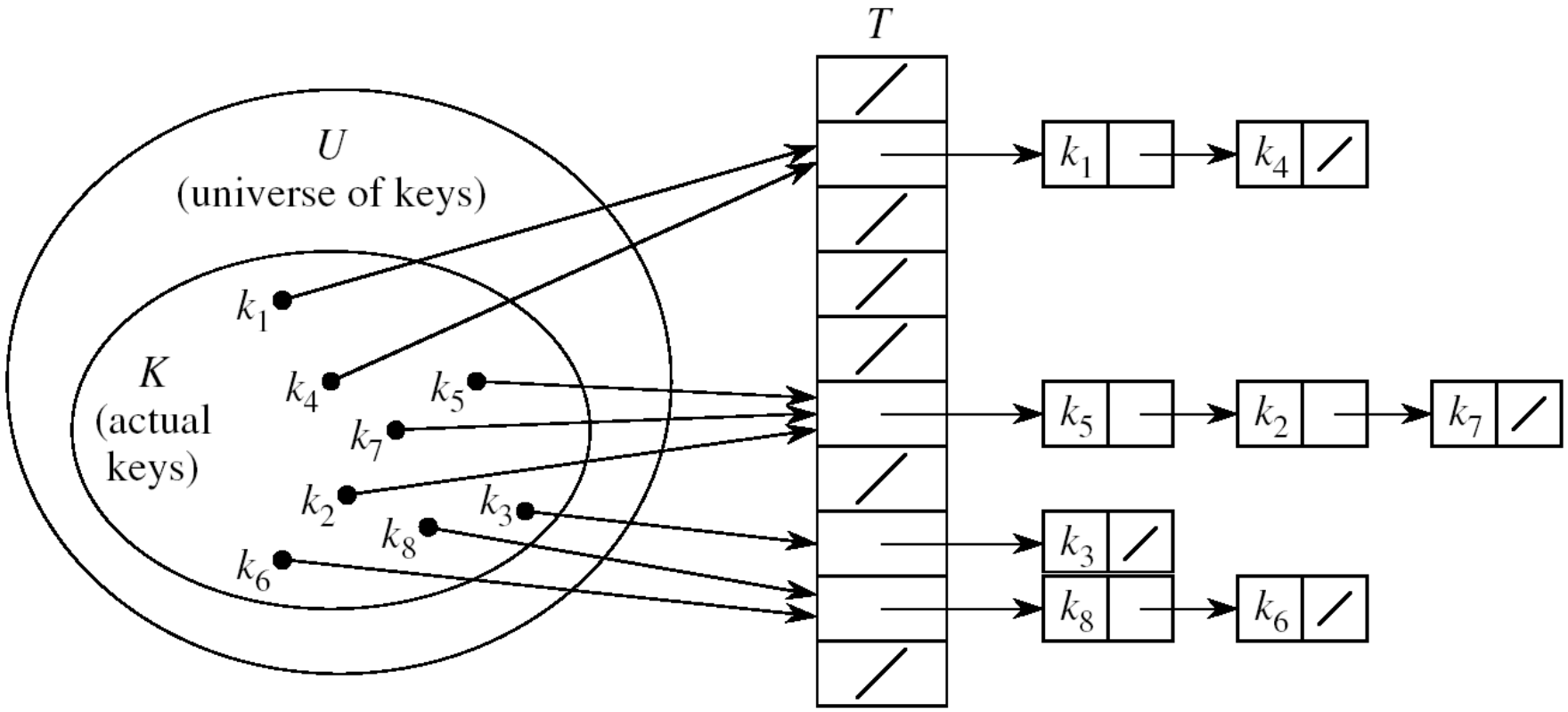
- Collision happens when Two or more keys hash to the same slot
- For a given set K of keys:
  - If $|K| \leq m$, collisions may or may not happen, depending on the hash function and the key distribution.
  - If $|K| > m$, collisions will definitely happen (i.e., there must be at least two keys that have the same hash value)
- Avoiding collisions completely is hard, even with a good hash function

Handling Collision

- There are multiple approaches to successfully handle collision:
  - Chaining
  - Open addressing:
    - Linear probing
    - Quadratic probing
    - Double hashing

Handling Collision Through Chaining

- The idea is to insert all elements that hash to the same slot into a linked list



Chaining Analysis

- Worst case search scenario is when all keys hash to the same slot, so it ends up a linked list with O(n).
- If we assume simple uniform distribution, then the probability for this to happen is: $1/m^n$ (why?)
- Expected length of a chain follows the expected value formula: $E|X| = \sum_{i=1}^{n} x_i p_i$, This follows Bernoulli distribution, and hence $= n/m$ which is known as load factor λ
- As long as we maintain m to be Θ(n), then λ is constant, and hence running time of search is O(1+ λ). 1 is for the hashing time.
- It is important then to choose the table size such that:
  - Small enough to avoid wasting a lot of space
  - Large enough such that chains (lists) remain short
  - Typically, 1/5 or 1/10 of the total number of elements.

Open Addressing

- Separate chaining requires using two data structures (array and linked list), which could impact the search speed.
- For m > n there could be a lot of space wasted in the array itself
- An alternative approach to resolve collisions is to attempt to find a contiguous empty cell to use as the key index.
- The cell allocation follows the following formula:
  - $h_i(x) = \big(hash(x) + f(i)\big)\ mod\ TableSize, with\ f(0) = 0$
  - f is the collision resolution strategy
- These tables (usually $\lambda < 0.5$) are called probing hash tables.
- There are three common collision resolution strategies

Linear Probing

- In linear probing, f is linear function of i, typically $f(i) = i$
- It tries sequentially to find an empty cell.
- For example, inserting {89, 18, 49, 58, and 69} in a probing hash table of size 10, and the hash function $h(k) = k \; mod \; m$

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	89

0	
1	
2	
3	
4	
5	
6	
7	
8	18
9	89

0	49
1	
2	
3	
4	
5	
6	
7	
8	18
9	89

0	49
1	58
2	
3	
4	
5	
6	
7	
8	18
9	89

0	49
1	58
2	69
3	
4	
5	
6	
7	
8	18
9	89

Linear Probing Issues

- Time to find an empty spot to insert can become a challenge
- With relatively empty array, blocks of occupied cells start to form, which is known as primary clustering
- Primary clustering means that any key that hashes into the cluster will require several attempts to solve the collision before adding to the cluster
- It can be shown that expected number of probes using linear probing is roughly $1/2(1 + 1/(1 - \lambda)^2)$ for insertion and unsuccessful search, and $1/2\big(1 + 1/(1 - \lambda)\big)$ for successful search

Quadratic Probing

- Quadratic probing is another collision resolution strategy that minimizes the clustering issue created by linear probing.
- f is quadratic and typically $f(i) = i^2$
- Repeating the same example {89, 18, 49, 58, and 69}, shows:

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	89

0	
1	
2	
3	
4	
5	
6	
7	
8	18
9	89

0	49
1	
2	
3	
4	
5	
6	
7	
8	18
9	89

0	49
1	
2	58
3	
4	
5	
6	
7	
8	18
9	89

0	49
1	
2	58
3	69
4	
5	
6	
7	
8	18
9	89

Quadratic Probing Issues

- Probing can go into cycles in attempt to find an empty spot, especially if the table size is not prime, and/or the table is half full or more.
- Theorem: If quadratic probing is used, and the table size is prime, then a new element can always be inserted if the table is at least half empty
- In other words, slots visited during the first M/2 probes (or iterations) are distinct.

Quadratic Theorem Proof

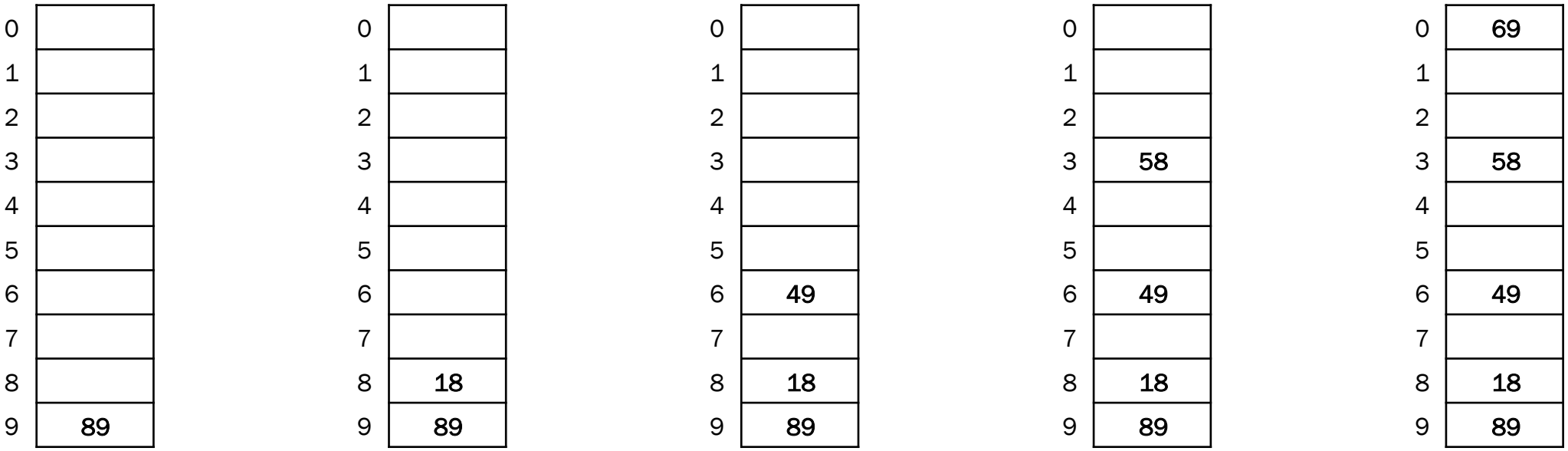
- Proof by contradiction:
  - Suppose there is a slot which is visited twice during the first $M/2$ iterations. Let i and j be the two iterations, where $0 \leq i < j \leq M/2$. Then:
  - $(H + i^2)\ mod\ M = (H + j^2)\ mod\ M$. Since i ≠ j, then one term must be multiple M's more of the other. Hence, there exist some c such that:
  - $(H + j^2) = (H + i^2) + cM$, since j is > i
  - $j^2 = i^2 + cM$ ➔ $j^2 - i^2 = cM$ ➔ $(\text{j} - i)(j + i) = cM$
  - Since M is prime, then it must be one of the terms divisible by M
  - Case 1: $(\text{j} - i)\ is\ divisble\ by\ M$
    - From the assumption, $i < j \leq M/2$ ➔ $(\text{j} - i) < M$ ➔ Contradiction
  - Case 2: $(\text{j} + i)\ is\ divisble\ by\ M$
    - From the assumption, $i < j \leq M/2$ ➔ $(\text{j} + i) < M$ ➔ Contradiction
- Hence, there are no two iterations which probe to the same slot during the first M/2 iterations

Double Hashing

- As the name suggests, two hashing functions are applied:
  - First hash function to determine the first slot a key should normally hash to
  - Second hash function to determine the increment for the probe sequence.
- h(k,i) = (h1(k) + i*h2(k)) mod m, i=0,1,…
- Selection of the second hash function is crucial:
  - The hash function must never evaluate to 0
  - All cells can be probed
- One good selection is: $h2(k) = R - (k \ mod \ R)$, R is a prime smaller than the table size

Double hashing

- Repeat the same example with double hashing, and $h2(k) = 7 - (k \bmod 7)$
  - Second hash function to determine the increment for the probe sequence.
- h(k,i) = (h1(k) + i*h2(k)) mod m, i=0,1,…



To Do List

- Review lecture notes
- Work on homework 2