

CS 372: Algorithms

Hash Tables

(Based on slides by David Luebke)

1

Review: Hash Tables

- Motivation: symbol tables
 - A compiler uses a *symbol table* to relate symbols to associated data
 - Symbols: variable names, procedure names, etc.
 - Associated data: memory location, call graph, etc.
 - For a symbol table (also called a *dictionary*), we care about search, insertion, and deletion
 - We typically don't care about sorted order

2

Review: Hash Tables

- More formally:
 - Given a table T and a record x , with key (= symbol) and satellite data, we need to support:
 - Insert (T, x)
 - Delete (T, x)
 - Search(T, x)
 - We want these to be fast, but don't care about sorting the records
- The structure we will use is a *hash table*
 - Supports all the above in $O(1)$ expected time!

3

Review: Direct Addressing

- Suppose:
 - The range of keys is $0..m-1$
 - Keys are distinct
- The idea:
 - Set up an array $T[0..m-1]$ in which
 - $T[i] = x$ if $x \in T$ and $\text{key}[x] = i$
 - $T[i] = \text{NULL}$ otherwise
 - This is called a *direct-address table*
 - Operations take $O(1)$ time!
 - *So what's the problem?*

4

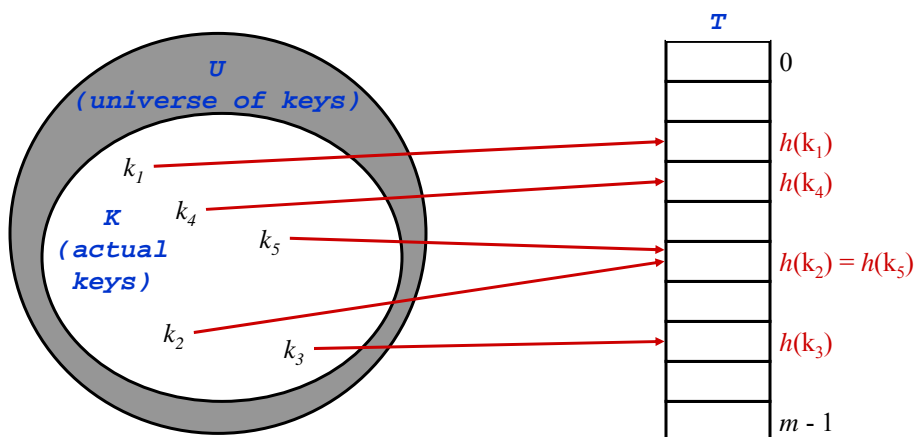
The Problem With Direct Addressing

- Direct addressing works well when the range m of keys is relatively small
- But what if the keys are 32-bit integers?
 - Problem 1: direct-address table will have 2^{32} entries, more than 4 billion
 - Problem 2: even if memory is not an issue, the time to initialize the elements to NULL may be
- Solution: map keys to smaller range $0..m-1$
- This mapping is called a *hash function*

5

Hash Functions

- Next problem: *collision*



6

Resolving Collisions

- *How can we solve the problem of collisions?*
- Solution 1: *chaining*
- Solution 2: *open addressing*

7

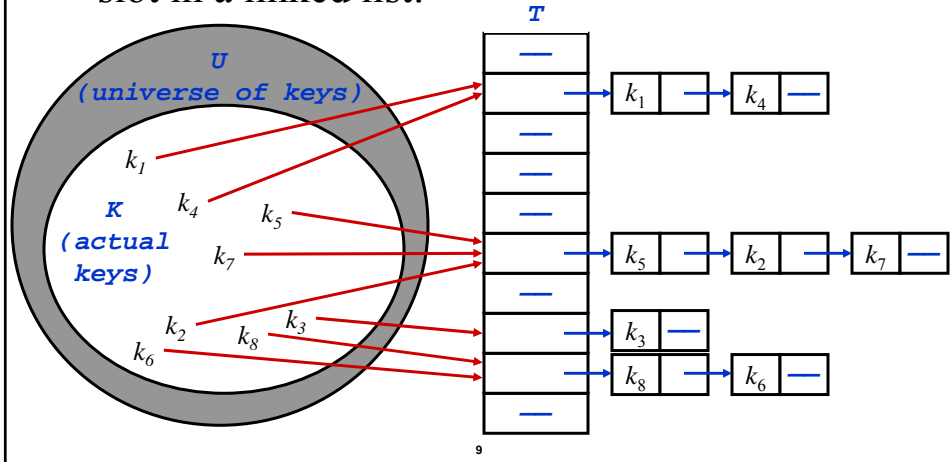
Open Addressing

- Basic idea (details in Section 11.4):
 - To insert: if slot is full, try another slot, ..., until an open slot is found (*probing*)
 - To search, follow same sequence of probes as would be used when inserting the element
 - If reach element with correct key, return it
 - If reach a NULL pointer, element is not in table
- Good for fixed sets (adding but no deletion)
 - Example: spell checking
- Table needn't be much bigger than n

8

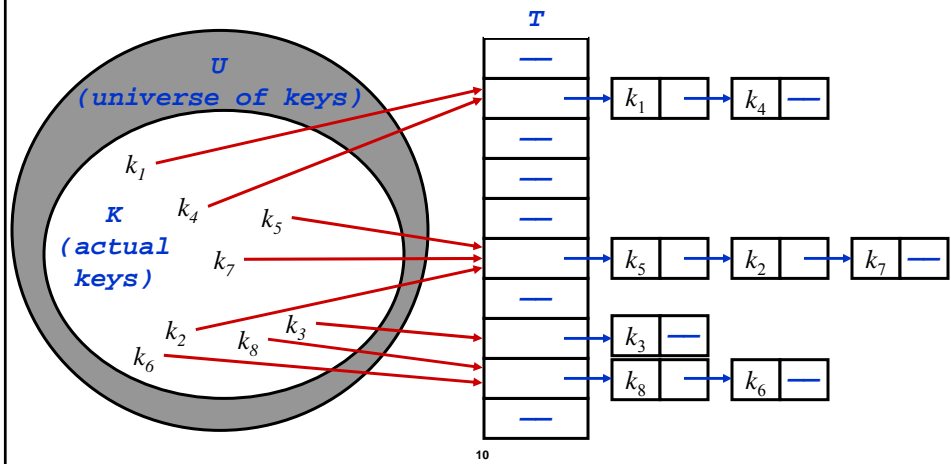
Chaining

- Chaining puts elements that hash to the same slot in a linked list:



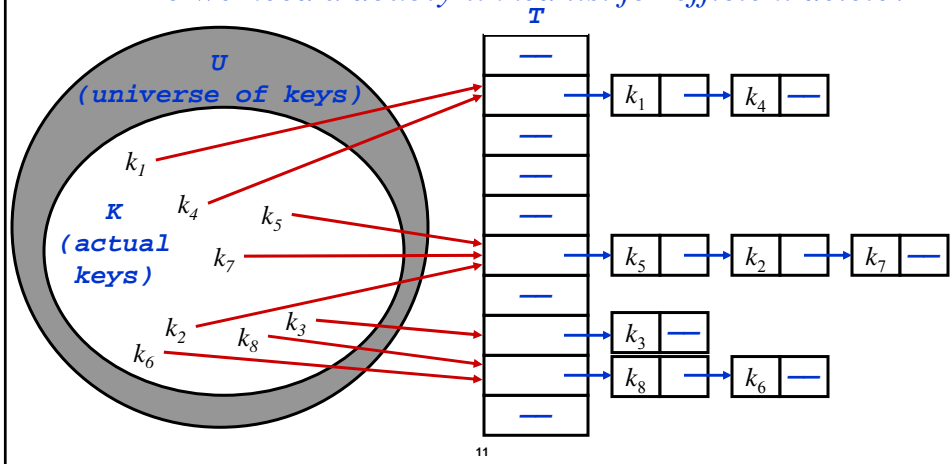
Chaining

- How do we insert an element?



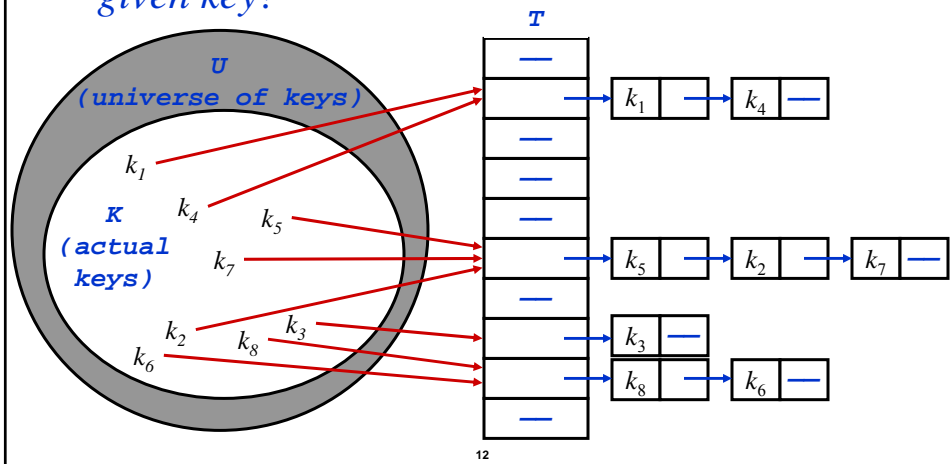
Chaining

- How do we delete an element?
 - Do we need a doubly-linked list for efficient delete?



Chaining

- How do we search for a element with a given key?



Analysis of Chaining

- Assume *simple uniform hashing*: each key in table is equally likely to be hashed to any slot
- Given n keys and m slots in the table: the *load factor* $\alpha = n/m =$ average # keys per slot
- *What will be the average cost of an unsuccessful search for a key?*

13

Analysis of Chaining

- Assume *simple uniform hashing*: each key in table is equally likely to be hashed to any slot
- Given n keys and m slots in the table, the *load factor* $\alpha = n/m =$ average # keys per slot
- *What will be the average cost of an unsuccessful search for a key?* A: $O(1+\alpha)$

14

Analysis of Chaining

- Assume *simple uniform hashing*: each key in table is equally likely to be hashed to any slot
- Given n keys and m slots in the table, the *load factor* $\alpha = n/m =$ average # keys per slot
- *What will be the average cost of an unsuccessful search for a key?* A: $O(1+\alpha)$
- *What will be the average cost of a successful search?*

15

Analysis of Chaining

- Assume *simple uniform hashing*: each key in table is equally likely to be hashed to any slot
- Given n keys and m slots in the table, the *load factor* $\alpha = n/m =$ average # keys per slot
- *What will be the average cost of an unsuccessful search for a key?* A: $O(1+\alpha)$
- *What will be the average cost of a successful search?* A: $O(1 + \alpha/2) = O(1 + \alpha)$

16

Analysis of Chaining Continued

- So the cost of searching = $O(1 + \alpha)$
- *If the number of keys n is proportional to the number of slots in the table, what is α ?*
- A: $\alpha = O(1)$
 - In other words, we can make the expected cost of searching constant if we make α constant

17

Choosing A Hash Function

- Clearly choosing the hash function well is crucial
 - *What will a worst-case hash function do?*
 - *What will be the time to search in this case?*
- *What are desirable features of the hash function?*
 - Should distribute keys uniformly into slots
 - Should not depend on patterns in the data

18

Hash Functions: The Division Method

- $h(k) = k \bmod m$
 - In words: hash k into a table with m slots using the slot given by the remainder of k divided by m
- *What happens to elements with adjacent values of k ?*
- *What if m is a power of 10?*
- *What happens if m is a power of 2 (say 2^p)?*
- Upshot: pick table size $m =$ prime number not too close to a power of 2 (or 10)

19

Hash Functions: The Multiplication Method

- For a constant A , $0 < A < 1$:
- $h(k) = \lfloor m (kA - \lfloor kA \rfloor) \rfloor$
What does this term represent?

20

Hash Functions: The Multiplication Method

- For a constant A , $0 < A < 1$:
- $h(k) = \lfloor m \underbrace{(kA - \lfloor kA \rfloor)}_{\text{Fractional part of } kA} \rfloor$
- Choose $m = 2^p$
- Choose A not too close to 0 or 1
- Knuth: Good choice for $A = (\sqrt{5} - 1)/2$

21

Hash Functions: Worst Case Scenario

- Scenario:
 - You are given an assignment to implement hashing
 - You will self-grade in pairs, testing and grading your partner's implementation
 - In a blatant violation of the honor code, your partner:
 - Analyzes your hash function
 - Picks a sequence of "worst-case" keys, causing your implementation to take $O(n)$ time to search
- *What's an honest CS student to do?*

22

Hash Functions: Universal Hashing

- As before, when attempting to foil an malicious adversary: randomize the algorithm
- *Universal hashing*: pick a hash function randomly in a way that is independent of the keys that are actually going to be stored
 - Guarantees good performance on average, no matter what keys adversary chooses