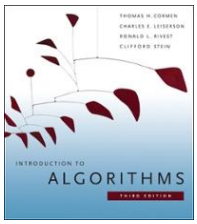
CS146 Data Structures and Algorithms



Chapter 11: Hash Tables

Slides prepared by Dr. Mike Wu of SJSU and Some adopted from Dr. David Lin of Virginia Tech.

Why? Hashing 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

Dictionary

- **Dictionary:**
  - Dynamic-set data structure for storing items indexed using *keys*.
  - Supports operations Insert, Search, and Delete.
  - Applications:
    - Symbol table of a compiler.
    - Memory-management tables in operating systems.
    - Predicting search keywords (Google search engine, etc.)
- **Hash Tables:**
  - Effective way of implementing dictionaries.
  - Generalization of ordinary arrays.

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!

Hashing: Keys

- In the following discussions we will consider all keys to be (possibly large) natural numbers
- *How can we convert floats to natural numbers for hashing purposes?*
- *How can we convert ASCII strings to natural numbers for hashing purposes?*

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 key[*x*] = *i*
    - T[*i*] = NULL otherwise
  - This is called a *direct-address table*
    - Operations take O(1) time!
    - *So what's the problem?*

Direct-address Tables

- Direct-address Tables are ordinary arrays.
- Facilitate direct addressing.
  - Element whose key is *k* is obtained by indexing into the k^{th} position of the array.
- Applicable when we can afford to allocate an array with one position for every possible key.
  - i.e. when the universe of keys *U* is small.
- Dictionary operations can be implemented to take *O*(1) time.
  - Details in Sec. 11.1.

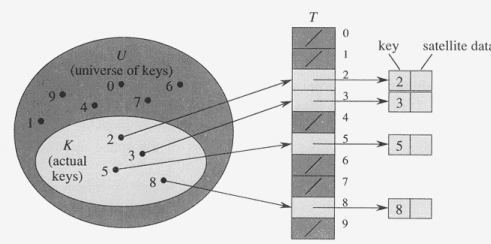


Figure 11.1 Implementing a dynamic set by a direct-address table T. Each key in the universe $U = \{0, 1, \ldots, 9\}$ corresponds to an index in the table. The set $K = \{2, 3, 5, 8\}$ of actual keys determines the slots in the table that contain pointers to elements. The other slots, heavily shaded, contain NIL.

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*