Computación y Estructuras Discretas I

Andrés A. Aristizábal P.
aaaristizabal@icesi.edu.co

Departamento de Computación y Sistemas Inteligentes

2024-2

Agenda

1 **Non recursive discrete structures**
- Reading control
- Dictionaries
- Introduction
- Direct-address tables
- Hash tables
- Hash functions

Reading control

Instructions

1. Go to https://www.socrative.com
2. Log in
3. Student Login
4. Room name: FTJF5SP
5. Name
6. Start

Agenda

1. **Non recursive discrete structures**
   - Reading control
   - Dictionaries
   - Introduction
   - Direct-address tables
   - Hash tables
   - Hash functions

Dictionaries

Intuitively, what is a Dictionary data structure?

Dictionaries

Intuitively, what is a Dictionary data structure?

- A data structure used to manipulate objects in which one periodically insert and extract elements.
- One can verify if an specific element belongs or not to the collection.

Dictionaries

Intuitively, what is a Dictionary data structure?

- A data structure used to manipulate objects in which one periodically insert and extract elements.
- One can verify if an specific element belongs or not to the collection.

How are Dictionaries also known as?

Dictionaries

Intuitively, what is a Dictionary data structure?

- A data structure used to manipulate objects in which one periodically insert and extract elements.
- One can verify if an specific element belongs or not to the collection.

How are Dictionaries also known as?

Associative arrays or maps

Dictionaries

What does each element in a dictionary have?

Dictionaries

What does each element in a dictionary have?

- Each element has:
  - A key
  - An associated value for that key
- The analogy with the real world dictionary comes from the fact that in every dictionary we have:
  - Words (keys)
  - Descriptions related to each word (value)

Dictionaries

What is the way in which a dictionary holds data?

Dictionaries

What is the way in which a dictionary holds data?

- By means of pairs
- (*key*, *value*)
- The data held in the structure are the values.
- The key is used for searching and finding the required values.

Dictionaries

In which everyday real world examples could we use dictionaries?

Dictionaries

What is the difference between an array and a dictionary?

Dictionaries

What is the difference between an array and a dictionary?

- In an array the key must be a number (a positive or non negative integer).
- In a dictionary the key can be any type of object.

Dictionaries

What is the difference between an array and a dictionary?

- In an array the key must be a number (a positive or non negative integer).
- In a dictionary the key can be any type of object.

Usually, what would these keys be?

Dictionaries

What is the difference between an array and a dictionary?

- In an array the key must be a number (a positive or non negative integer).
- In a dictionary the key can be any type of object.

Usually, what would these keys be?

- A random set of values such as real numbers or strings.
- With the restriction that each key must be distinguishable from the other.

Dictionaries

What are the required operations defined by this data structure?

Dictionaries

What are the required operations defined by this data structure?

- `void Add(K key, V value)`
- `V Get(K key)`
- `boolean Remove(K key)`

Dictionaries

And some additional methods?

Dictionaries

And some additional methods?

- `boolean Contains(K key)`
- `int Count()`

Agenda

1 **Non recursive discrete structures**
- Reading control
- Dictionaries
- Introduction
- Direct-address tables
- Hash tables
- Hash functions

Introduction

Why is the implementation of dictionary operations important?

Introduction

Why is the implementation of dictionary operations important?

Many applications require a dynamic set that supports only the dictionary operations INSERT, SEARCH, and DELETE.

Introduction

Why is the implementation of dictionary operations important?

Many applications require a dynamic set that supports only the dictionary operations INSERT, SEARCH, and DELETE.

Such as?

Introduction

Why is the implementation of dictionary operations important?

Many applications require a dynamic set that supports only the dictionary operations INSERT, SEARCH, and DELETE.

Such as?

- A compiler (translates a programming language)
    - Maintains a symbol table.
    - In which the keys of elements are arbitrary character strings corresponding to identifiers in the language.

Introduction

What is an effective way of implementing dictionaries?

Introduction

What is an effective way of implementing dictionaries?

Using hash tables.

Introduction

What is an effective way of implementing dictionaries?

Using hash tables.

Why?

Introduction

What is an effective way of implementing dictionaries?

Using hash tables.

Why?

- Although searching for an element in a hash table can take $\Theta(n)$ time in the worst case, in practice, hashing performs extremely well.
- Under reasonable assumptions, the average time to search for an element in a hash table is $O(1)$.

Introduction

What does a hash table do?

Introduction

What does a hash table do?

- Generalizes the simpler notion of an ordinary array.
- Directly addressing into an ordinary array makes effective use of our ability to examine an arbitrary position in an array in $O(1)$ time.
- We can take advantage of direct addressing when we can afford to allocate an array that has one position for every possible key.
- When the number of keys actually stored is small relative to the total number of possible keys, hash tables become an effective alternative to directly addressing an array.

Agenda

1 **Non recursive discrete structures**
- Reading control
- Dictionaries
- Introduction
- Direct-address tables
- Hash tables
- Hash functions

Direct-address tables

What is direct addressing?

Direct-address tables

What is direct addressing?

- It is a simple technique that works well when the universe U of keys is reasonably small.
- Suppose that an application needs a dynamic set in which each element has a key drawn from the universe $U = \{0, 1, ..., m - 1\}$ where m is not too large and no two elements have the same key.
    - To represent the dynamic set, we use an array, or direct-address table, denoted by $T[0..m - 1]$ in which each position, or slot, corresponds to a key in the universe U.
    - Slot k points to an element in the set with key k
    - If the set contains no element with key k, then $T[k] = NIL$

Direct-address tables

How would the implementation of dictionary operations using direct addressing be?

Direct-address tables

How would the implementation of dictionary operations using direct addressing be?

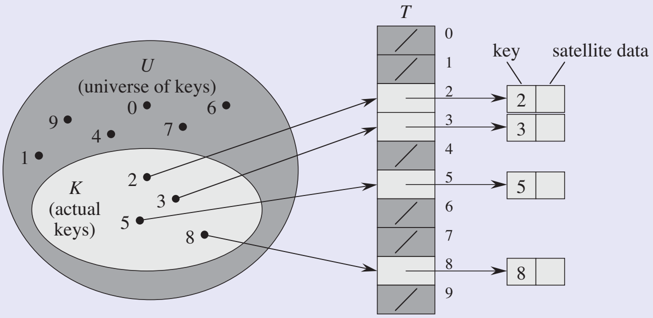
```
DIRECT−ADDRESS−SEARCH(T,k)
  return T[k]

DIRECT−ADDRESS−INSERT(T,x)
  T[x.key] = x

DIRECT−ADDRESS−DELETE(T,x)
  T[x.key] = NIL
```

Each of these operations takes only $O(1)$.

Direct-address tables

Example

Given $U = \{0, 1, ..., 9\}$ *and* $K = \{2, 3, 5, 8\}$



Agenda

1 **Non recursive discrete structures**
- Reading control
- Dictionaries
- Introduction
- Direct-address tables
- Hash tables
- Hash functions

Hash tables

What is the obvious downside of using direct addressing?

Hash tables

What is the obvious downside of using direct addressing?

- If the universe U is large, storing a table T of size $|U|$ may be impractical, or even impossible, given the memory available on a typical computer.
- If $|K| << |U|$ most of the space allocated for T would be wasted.

Hash tables

What to do then if $|K| << |U|$?

Hash tables

What to do then if $|K| << |U|$?

- Using a hash table since it requires much less storage than a direct-address table.
- We can reduce the storage requirement to $\Theta(|K|)$.
- We maintain the benefit that searching for an element in the hash table still requires only $O(1)$ time
- Unfortunately it takes $O(1)$ for the average-case time and not for the worst-case time.

Hash tables

What is hashing?

Hash tables

What is hashing?

- While with direct addressing, an element with key k is stored in slot k, with hashing, this element is stored in slot $h(k)$. I.e. we use a hash function h to compute the slot from the key k.
- Here, h maps the universe U of keys into the slots of a hash table $T[0..m-1]$

Hash tables

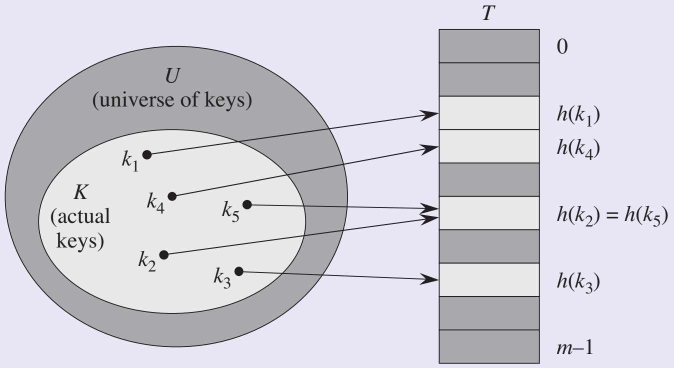
$$h : U \rightarrow \{0, 1, ..., m - 1\}$$

Hash tables

$$h : U \to \{0, 1, ..., m-1\}$$

- Where $m << |U|$ and each element has a different key.
- $h(k)$ is the hash value of key k.
- The hash function reduces the range of array indices and hence the size of the array.
- Instead of a size of $|U|$, the array can have size m.

Hash tables



U
(universe of keys)
K
(actual keys)
k_1
k_4
k_5
k_2
k_3
T
0
$h(k_1)$
$h(k_4)$
$h(k_2) = h(k_5)$
$h(k_3)$
$m-1$

Hash tables

What is the problem with this solution?

Hash tables

What is the problem with this solution?

- Two keys may hash to the same slot.
- We call this situation a collision.

Hash tables

What is the problem with this solution?

- Two keys may hash to the same slot.
- We call this situation a collision.

How to solve the problem with collisions?

Hash tables

What is the problem with this solution?

- Two keys may hash to the same slot.
- We call this situation a collision.

How to solve the problem with collisions?

- The ideal solution would be to avoid collisions altogether.
  - We might try to achieve this goal by choosing a suitable hash function h.
  - Make h appear to be random, thus avoiding collisions or at least minimizing their number.
  - Because $|U| > m$, however, there must be at least two keys that have the same hash value; avoiding collisions altogether is therefore impossible.

Hash tables

What is another way of solving this problem?

Hash tables

What is another way of solving this problem?

By means of collision resolution techniques such as chaining.

Hash tables

What is another way of solving this problem?

By means of collision resolution techniques such as chaining.

What is chaining?

Hash tables

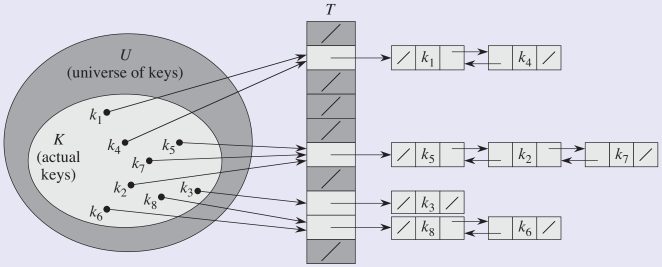
What is another way of solving this problem?

By means of collision resolution techniques such as chaining.

What is chaining?

- Is the mechanism by which we place all the elements that hash to the same slot into the same linked list.
- Slot j contains a pointer to the head of the list of all stored elements that hash to j.
- If there are no such elements, slot j contains NIL.

Hash tables



T
U
(universe of keys)
K
(actual keys)
k_1
k_4
k_5
k_7
k_2
k_3
k_8
k_6
k_1
k_4
k_5
k_2
k_7
k_3
k_8
k_6

Hash tables

How would the implementation of dictionary operations using hash tables and chaining as a collision resolution technique be?

Hash tables

How would the implementation of dictionary operations using hash tables and chaining as a collision resolution technique be?

```
CHAINED−HASH−INSERT(T,x)
  insert x at the head of list T[h(x.key)]

CHAINED−HASH−SEARCH(T,k)
  search for an element with key k in list T[h(k)]

CHAINED−HASH−DELETE(T,x)
  delete x from the list T[h(x.key)]
```

Hash tables

Which would be the execution times for each of the previous operations?

Hash tables

Which would be the execution times for each of the previous operations?

- The worst-case running time for insertion is $O(1)$
- For searching, the worst- case running time is proportional to the length of the list.
- We can delete an element in $O(1)$ time if the lists are doubly linked (Note that CHAINED-HASH-DELETE takes as input an element x and not its key k, so that we don't have to search for x first. If the hash table supports deletion, then its linked lists should be doubly linked so that we can delete an item quickly).

Hash tables

How long would it take to search for an element with a given key using chaining as a collision resolution technique?

Hash tables

How long would it take to search for an element with a given key using chaining as a collision resolution technique?

- Given a hash table T with m slots that stores n elements, we define the load factor α for T as n/m.
- The load factor indicates the average number of elements stored in a chain.
- The worst-case behavior of hashing with chaining is terrible.
    - All n keys hash to the same slot, creating a list of length n.
    - The worst-case time for searching is thus $\Theta(n)$, plus the time to compute the hash function.
    - Clearly, we do not use hash tables for their worst-case performance.

Hash tables

- The average-case performance of hashing depends on how well the hash function h distributes the set of keys to be stored among the m slots, on the average.
- We assume simple uniform hashing.
  - Any given element is equally likely to hash into any of the m slots, independently of where any other element has hashed to.
- We set aside the $O(1)$ time required to compute the hash function and to access slot $h(k)$.
- We consider the expected number of elements examined by the search algorithm, that is, the number of elements in the list $T[h(k)]$ that the algorithm checks to see whether any have a key equal to k.

Hash tables

- We shall consider two cases:
  - In the first, the search is unsuccessful: no element in the table has key k.
  - In the second, the search successfully finds an element with key k.

Hash tables

Theorem 1

In a hash table in which collisions are resolved by chaining, an unsuccessful search takes average-case time $\Theta(1 + \alpha)$ under the assumption of simple uniform hashing.

Hash tables

Theorem 1

In a hash table in which collisions are resolved by chaining, an unsuccessful search takes average-case time $\Theta(1 + \alpha)$ under the assumption of simple uniform hashing.

Theorem 2

In a hash table in which collisions are resolved by chaining, a successful search takes average-case time $\Theta(1 + \alpha)$ under the assumption of simple uniform hashing.

Hash tables

What does this analysis mean?

Hash tables

What does this analysis mean?

- If the number of hash-table slots is at least proportional to the number of elements in the table, we have $n = O(m)$.
- Consequently $\alpha = n/m = O(m)/m = O(1)$.
- Searching takes constant time on average.
- We can support all dictionary operations in $O(1)$ time on average.

Agenda

1 **Non recursive discrete structures**
- Reading control
- Dictionaries
- Introduction
- Direct-address tables
- Hash tables
- Hash functions

Hash functions

What makes a good hash function?

Hash functions

What makes a good hash function?

- That satisfies (approximately) the assumption of simple uniform hashing.
- I.e. that each key is equally likely to hash to any of the *m* slots, independently of where any other key has hashed to
- Unfortunately, we typically have no way to check this condition.
- We rarely know the probability distribution from which the keys are drawn.

Hash functions

Example

- *It is quite common in a program to have similar identifier names such as, var1, var2, for example.*
- *A good hash function should assign them to different slots.*
- *In that way we can see independence between each pair of keys.*

Hash functions

Example

- *Occasionally we do know the distribution.*
- *If we know that the keys are random real numbers k independently and uniformly distributed in the range* $0 \leq k < 1$, *then the hash function*

  $$h(k) = \lfloor km \rfloor$$

  satisfies the condition of simple uniform hashing.

Hash functions

What can we do if the keys are strings?

Hash functions

What can we do if the keys are strings?

- Most hash functions assume that the universe of keys is the set $\mathbb{N} = \{0, 1, ...\}$.
- If the keys are not natural numbers, we find a way to interpret them as natural numbers.
- To interpret a String we can use the ASCII character set in which we consider characters between 0 and 127.
  - We might interpret the identifier `pt` as the pair of decimal integers (112,116) since $p = 112$ and $t = 116$ in ASCII
  - Finally we express it as a Radix-128 integer such that $pt = 112 \times 128^1 + 116 \times 128^0 = 14452$.

Hash functions

Which are some methods to create hash functions?

Hash functions

Which are some methods to create hash functions?

- Division method
  - $h(k) = k \ mod \ m$
  - In this cases m shouldn't be a power of 2.
  - if $m = 2^p$, then $h(k)$ is just the p lowest-order bits of k.
  - A prime not too close to an exact power of 2 is often a good choice for m.

Hash functions

Example

Let $m = 2^3 = 8$, *then* $h(k)$ *is just the* 3 *lowest-order bits of* k.

- $k = 0, \longrightarrow 0 \, mod \, 8 = 0 \longrightarrow \underline{000}$
- $k = 8, \longrightarrow 8 \, mod \, 8 = 0 \longrightarrow 1\underline{000}$
- $k = 16, \longrightarrow 16 \, mod \, 8 = 0 \longrightarrow 10\underline{000}$
- $k = 24, \longrightarrow 24 \, mod \, 8 = 0 \longrightarrow 11\underline{000}$
- $k = 4, \longrightarrow 4 \, mod \, 8 = 4 \longrightarrow \underline{100}$
- $k = 12, \longrightarrow 12 \, mod \, 8 = 4 \longrightarrow 1\underline{100}$
- $k = 20, \longrightarrow 20 \, mod \, 8 = 4 \longrightarrow 10\underline{100}$
- $k = 28, \longrightarrow 28 \, mod \, 8 = 4 \longrightarrow 11\underline{100}$

Hash functions

- Multiplication method
  - $h(k) = \lfloor m \times (k\,A\ mod\ 1) \rfloor$
  - Where $k\,A\ mod$ 1 is the fractional part of $k\,A$. $(k\,A - \lfloor k\,A \rfloor)$
  - $0 < A < 1$.

Hash functions

- The value of m is not critical.
- We do not need to avoid values of m as with the division method.
- We typically choose m to be a power of 2 ($m = 2^p$ for some integer p)
- Simpler computations.

Hash functions

- Although this method works with any value of the constant A, it works better with some values than with others.
- The optimal choice depends on the characteristics of the data being hashed.
- Knuth suggests that $A \approx (\sqrt{5} - 1)/2 = 0{,}6180339887...$

Hash functions

- Universal hashing
  - In universal hashing, at the beginning of execution we select the hash function at random from a carefully designed class of functions.
  - Let $\mathcal{H} = \{h_1, h_2, ..., h_l\}$ be a finite collection of hash functions that map a given universe U of keys into the range $\{0, 1, ..., m - 1\}$
  - Such a collection is said to be universal if for each pair of distinct keys $x, y \in U$, the number of hash functions $h \in \mathcal{H}$ for which $h(x) = h(y)$ is at most $|\mathcal{H}|/m$
  - In other words, with a hash function randomly chosen from $\mathcal{H}$ the chance of a collision between distinct keys x, y is no more than the chance $1/m$ of a collision if $h(x)$ and $h(y)$ were randomly and independently chosen from the set $\{0, 1, ..., m - 1\}$