CSCI 2270: Data Structures

Lecture 21: Hash Tables

Ashutosh Trivedi

Key1	Value1
Key2	Value2
Key3	Value3

Department of Computer Science
UNIVERSITY OF COLORADO BOULDER

Dictionary

Definition.

1. *A book or electronic resource that lists the words of a language (typically in alphabetical order) and gives their meaning.*
2. *A reference work on a particular subject, the items of which are typically arranged in alphabetical order.*

Dictionary

Definition.

1. *A book or electronic resource that lists the words of a language (typically in alphabetical order) and gives their meaning.*
2. *A reference work on a particular subject, the items of which are typically arranged in alphabetical order.*

Data Structure.

- A general-purpose data structure for storing a group of objects.
- A set of keys and each key has a single associated value.
- Given a key, the dictionary will return the associated value.
- Also known as associative array, map, symbol table.
- Example:
  - Students and their grades,
  - book titles and shelf number,
  - movie title and other information, etc.
- Three key operations are INSERT, DELETE, and SEARCH.
- Can be implemented as an array (sorted or unsorted), as a linked list, or as a binary search tree.

Motivation

How efficiently we can perform search?

Motivation

How efficiently we can perform search?

What is the complexity of the following dictionary operations when implemented as an array, sorted-array, linked-list, binary search tree?

Motivation

How efficiently we can perform search?

What is the complexity of the following dictionary operations when implemented as an array, sorted-array, linked-list, binary search tree?

	Array	Sorted-array	Linked Lists	BST
INSERT	$O(1)$	$O(n)$	$O(1)$	$O(\log(n))$
DELETE	$O(n)$	$O(n)$	$O(n)$	$O(\log(n))$
SEARCH	$O(n)$	$O(\log(n))$	$O(n)$	$O(\log(n))$

Motivation

How efficiently we can perform search?

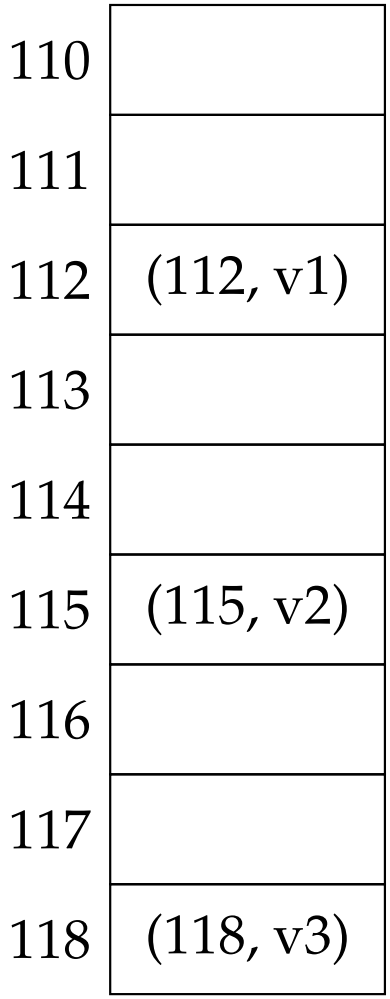
What is the complexity of the following dictionary operations when implemented as an array, sorted-array, linked-list, binary search tree?

	Array	Sorted-array	Linked Lists	BST
INSERT	$O(1)$	$O(n)$	$O(1)$	$O(\log(n))$
DELETE	$O(n)$	$O(n)$	$O(n)$	$O(\log(n))$
SEARCH	$O(n)$	$O(\log(n))$	$O(n)$	$O(\log(n))$

Is it possible to get insert, delete, and search operations with $O(1)$ complexity?

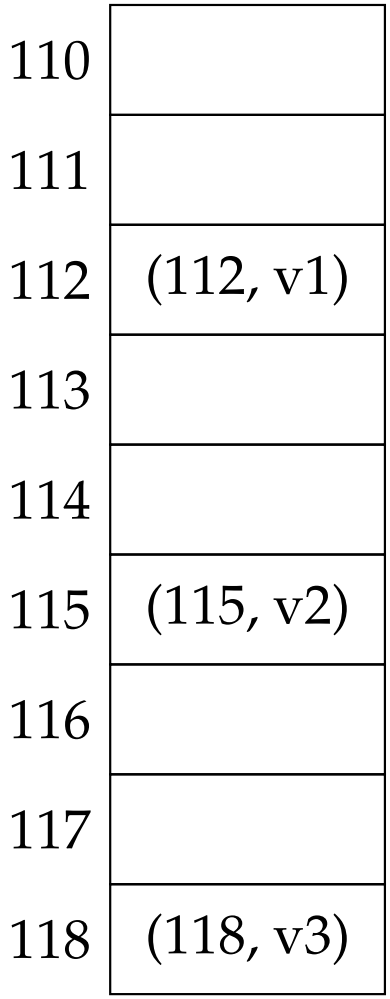
Direct-Address Tables

When the universe of key values is small:



Direct-Address Tables

When the universe of key values is small:



INSERT : $O(1)$, DELETE : $O(1)$, and SEARCH : $O(1)$.

Storage Requirements

What is the problem with this approach?

Storage Requirements

What is the problem with this approach?

When number of values to store is quite small, but the universe of key values is large, then this approach wastes a lot of memory.

Storage Requirements

What is the problem with this approach?

When number of values to store is quite small, but the universe of key values is large, then this approach wastes a lot of memory.

Solution. Hash-Tables!

Hash Tables

- A hash table is a data structure that stores data using a parameter in the data, called a key, to map the data to an index in an array.

Hash Tables

- A hash table is a data structure that stores data using a parameter in the data, called a key, to map the data to an index in an array.
- Hash-tables use a "hash function" to squish large range of key-values to a small range.
- Examples of hash functions:
  - `hash(int key) = key % 20`
  - `hash(string key) = sum-of-ASCII-values % 50`

Hash Tables

- A hash table is a data structure that stores data using a parameter in the data, called a key, to map the data to an index in an array.
- Hash-tables use a "hash function" to squish large range of key-values to a small range.
- Examples of hash functions:
  - `hash(int key) = key % 20`
  - `hash(string key) = sum-of-ASCII-values % 50`
- Instead of storing the key values directly at their index, hash-tables stores them at the index of their hashed representation.

Hash Tables

- A hash table is a data structure that stores data using a parameter in the data, called a key, to map the data to an index in an array.
- Hash-tables use a "hash function" to squish large range of key-values to a small range.
- Examples of hash functions:
  - `hash(int key) = key % 20`
  - `hash(string key) = sum-of-ASCII-values % 50`
- Instead of storing the key values directly at their index, hash-tables stores them at the index of their hashed representation.
- Hash collision:

  two distinct key values mapping to a same hashed value!
- We say that a has function is perfect when it assigns all records to a location in the hash table without collisions or wasted space.

Hash Tables

- A hash table is a data structure that stores data using a parameter in the data, called a key, to map the data to an index in an array.
- Hash-tables use a "hash function" to squish large range of key-values to a small range.
- Examples of hash functions:
  - `hash(int key) = key % 20`
  - `hash(string key) = sum-of-ASCII-values % 50`
- Instead of storing the key values directly at their index, hash-tables stores them at the index of their hashed representation.
- Hash collision:

  two distinct key values mapping to a same hashed value!
- We say that a has function is perfect when it assigns all records to a location in the hash table without collisions or wasted space.
- When perfect, it gives the same advantages of direct-address tables, with modest memory requirements.

Perfect/Imperfect Hash functions

Design a hash function to store up to 100 phone numbers in the range 7207079600 and 7207079699.

- a perfect hash function (k%100)

0	
1	
2	7207079602
3	
4	
5	7207079605
6	
7	
8	7207079608
9	

Perfect/Imperfect Hash functions

Design a hash function to store up to 100 phone numbers in the range 7207079600 and 7207079699.

- an imperfect hash function ($k\%10$)
- Collision Resolution: (I) open addressing (II) chaining

Perfect/Imperfect Hash functions

Design a hash function to store up to 100 phone numbers in the range 7207079600 and 7207079699.

- an imperfect hash function ($k\%10$)
- Collision Resolution: (I) open addressing (II) chaining

0	
1	
2	7207079602
3	7207079612
4	7207079622
5	7207079605
6	7207079655
7	
8	7207079608
9	

Perfect/Imperfect Hash functions

Design a hash function to store up to 100 phone numbers in the range 7207079600 and 7207079699.

- an imperfect hash function ($k\%10$)
- Collision Resolution: (I) open addressing (II) chaining

0	
1	
2	7207079602
3	7207079612
4	7207079622
5	7207079605
6	7207079655
7	
8	7207079608
9	

0	$\rightarrow$
1	$\rightarrow$
2	7207079602 $\rightarrow$ 7207079612 $\rightarrow$ 7207079622
3	$\rightarrow$
4	$\rightarrow$
5	7207079605 $\rightarrow$ 7207079655
6	$\rightarrow$
7	$\rightarrow$
8	7207079608 $\rightarrow$
9	$\rightarrow$

Open Addressing

In case of a hash-collision:

0	
1	
2	7207079602
3	7207079612
4	7207079622
5	7207079605
6	7207079655
7	
8	7207079608
9	

- Linear probing:
  - find the next free position in the array in the linear order, i.e. $h(x), h(x)+1, h(x)+2$

Open Addressing

In case of a hash-collision:

0	
1	
2	7207079602
3	7207079612
4	7207079622
5	7207079605
6	7207079655
7	
8	7207079608
9	

- Linear probing:
  - find the next free position in the array in the linear order, i.e. $h(x), h(x)+1, h(x)+2$
- Quadratic probing:
  - find the next free position in the array in the quadratic order, i.e. $h(x), h(x)+1^2, h(x)+2^2$

Open Addressing

In case of a hash-collision:

0	
1	
2	7207079602
3	7207079612
4	7207079622
5	7207079605
6	7207079655
7	
8	7207079608
9	

- Linear probing:
  - find the next free position in the array in the linear order, i.e. $h(x), h(x)+1, h(x)+2$
- Quadratic probing:
  - find the next free position in the array in the quadratic order, i.e. $h(x), h(x)+1^2, h(x)+2^2$
- Double hashing:
  - find the next free position in the array using double hash functions, i.e. $h_1(x) + i \cdot h_2(x)$.

Open Addressing

In case of a hash-collision:

0	
1	
2	7207079602
3	7207079612
4	7207079622
5	7207079605
6	7207079655
7	
8	7207079608
9	

- Linear probing:
  - find the next free position in the array in the linear order, i.e. $h(x), h(x)+1, h(x)+2$
- Quadratic probing:
  - find the next free position in the array in the quadratic order, i.e. $h(x), h(x)+1^2, h(x)+2^2$
- Double hashing:
  - find the next free position in the array using double hash functions, i.e. $h_1(x) + i \cdot h_2(x)$.
- Cuckoo hashing, Hopscotch hashing, Robin-hood hashing, and many more.