Dictionaries

Lecture 1: Intro, implementation using Arrays and Linked Lists

Definition

Dictionary is an abstract data structure that supports the following operations:

- search(K key)
  (returns the value associated with the given key)[1]
- insert(K key, V value)
- delete(K key)

Each element stored in a dictionary is identified by a key of type K. Dictionary represents a mapping from keys to values.

Dictionaries have numerous applications

[1] Search can return a special value if key is absent in dictionary

Examples

- contact book
  key: name of person; value: telephone number
- table of program variable identifiers
  key: identifier; value: address in memory
- property-value collection
  key: property name; value: associated value
- natural language dictionary
  key: word in language X; value: word in language Y
- etc.

Implementations

- simple implementations: sorted or unsorted sequences, direct addressing
- hash tables
- binary search trees (BST)
- AVL trees
- self-organising BST
- red-black trees
- (a,b)-trees (in particular: 2-3-trees)
- B-trees
- and other ...

Understanding the Data Type

Consider an empty unordered dictionary and the following set of operations:

Operation	Dictionary	Output
insertItem(5,A)	{(5,A)}	
insertItem(7,B)	{(5,A), (7,B)}	
insertItem(2,C)	{(5,A), (7,B), (2,C)}	
insertItem(8,D)	{(5,A), (7,B), (2,C), (8,D)}	
insertItem(2,E)	{(5,A), (7,B), (2,C), (8,D), (2,E)}	
findItem(7)	{(5,A), (7,B), (2,C), (8,D), (2,E)}	B
findItem(4)	{(5,A), (7,B), (2,C), (8,D), (2,E)}	NO_SUCH_KEY
findItem(2)	{(5,A), (7,B), (2,C), (8,D), (2,E)}	C
findAllItems(2)	{(5,A), (7,B), (2,C), (8,D), (2,E)}	C, E
size()	{(5,A), (7,B), (2,C), (8,D), (2,E)}	5
removeItem(5)	{(7,B), (2,C), (8,D), (2,E)}	A
removeAllItems(2)	{(7,B), (8,D)}	C, E
findItem(4)	{(7,B), (8,D)}	NO_SUCH_KEY

	Insertion	Removal	Retrieval	Traversal
Unsorted array-based	O(1)	O(*n*)	O(*n*)	O(*n*)
Unsorted link-based	O(1)	O(*n*)	O(*n*)	O(*n*)
Sorted array-based	O(*n*)	O(*n*)	O(log *n*)	O(*n*)
Sorted link-based	O(*n*)	O(*n*)	O(*n*)	O(*n*)
Binary search tree	O(log *n*)	O(log *n*)	O(log *n*)	O(*n*)