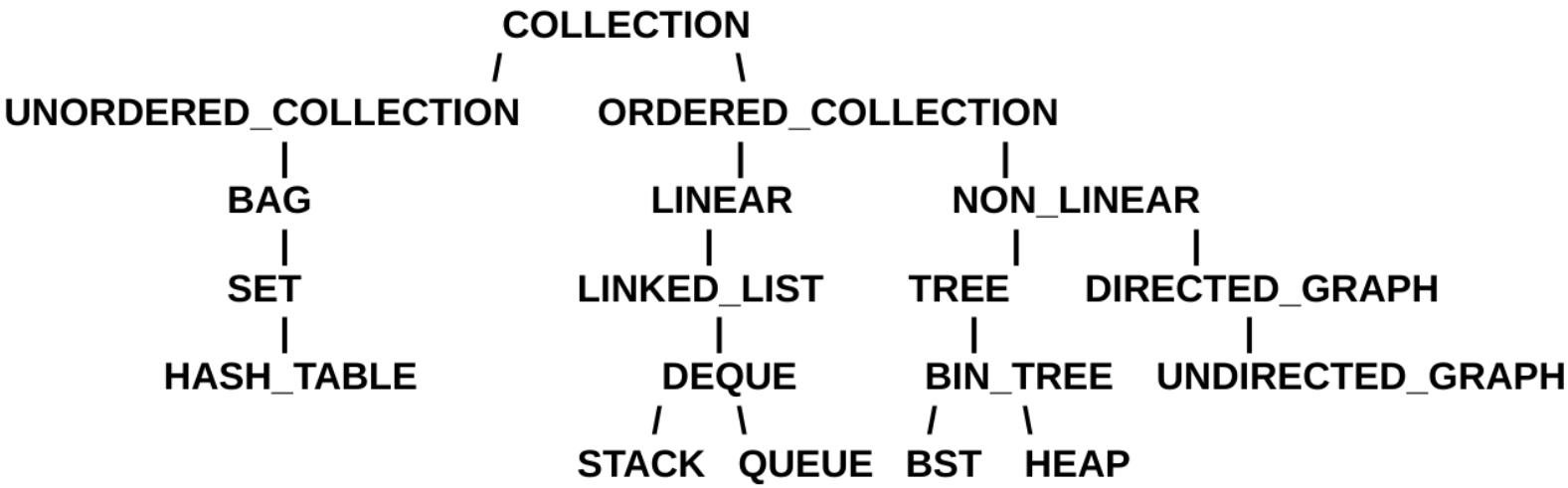
Associative array (a.k.a. dictionary, map)

Comp Sci 1575 Data Structures

Big picture on ADTs

"Yes, some books come in high definition — dictionaries!"

Big picture on ADTs



Interfaces used to create a dictionary

- List
- **Unsorted set (USet), and UMultiset**
- **Sorted set (SSet), and SMultiset**
- Priority queue
- Graph

Outline

Labeled data storage

- Dictionary is a classic computer science problem: the task of designing a data structure that maintains a set of data during `search`, `delete`, and `insert` operations.
- How can we efficiently organize collections of data records so that they can be stored and retrieved quickly?

Outline

Sets and associative arrays

Set: an un-ordered collection of non-repeating elements

Associative array: also known as a map, symbol table, or dictionary, is an abstract data type composed of a **set** of $< key, value >$ pairs, such that each possible key appears at most once in the collection. Association between a key and a value is often known as a "binding". Operations associated with this data type generally allow:

- **addition** of a pair to the collection
- **removal** of a pair from the collection
- **modification** of an existing pair
- **lookup** of a value associated with a particular key

Outline

Comparable keys and values

Key and Value

- **Search key:** comparable type to be searched for
- **Value:** the actual data record

Define **comparable**:

- Minimally, capable of taking two keys determining whether they are **equal or not**, which enables sequential search through a database of records to find one that matches a given key.
- **Total order** is sometimes better. For some data structures, you can determine which of two keys is greater than the other, which enables efficient search. Many simple data types have natural total orders: e.g., integers, floats, doubles, and character strings are all totally ordered.

Outline

Operations

- **Add or insert:** add a new (key, value) pair to the collection, binding the new key to its new value. *Arguments*: key and the value.
- **Remove or delete:** remove a (key, value) pair from the collection, unbinding a given key from its value. *Arguments*: key.
- **Reassign:** replace the value in one of the (key, value) pairs that are already in the collection, binding an old key to a new value. *Arguments*: key and the value.
- **Lookup or find:** find the value (if any) that is bound to a given key. *Arguments*: key. Returns the value from the operation. If no value is found, some associative array implementations raise an exception.

Alternatively, a single **set** operation that adds a new (key, value) pair if one does not already exist, and otherwise reassigns it. Also, often an **iterator** to loop over all bindings in an arbitrary pseudo-random order.

Outline

Types of dictionary data structures

1. **Sequential container** (e.g., association list) data structure can be used for dictionaries with a small number of bindings. To find the value associated with a given key, sequential search is used: each element of the list is searched, starting at the front, until the key is found. Time to perform the basic dictionary operations is linear in the total number of bindings, or $\log n$ in a sorted linear container
2. **Search trees** (coming up later)
3. **Hash table** is the most common (coming up later)

Outline

KQ and sketchpad

KVPair.h and dictionary.h

Check out the code

Dictionary implementations

Check out the code:
ualdict, salist, saldict

Outline

1 Introduction
Definitions
2 Comparable keys
Operations
3 Implementation
4 KQ and sketchpad
5 KQ and sketchpad

KQ and sketchpad

Asymptotic comparison of dictionary DS options

ADT	Lookup		Insertion		Deletion		Ordered
	Average	Worst	Average	Worst	Average	Worst	
Sequential container: key-value pairs	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	No
Sequential container: key-value pairs	O(log n)	O(n)	O(1)	O(1)	O(n)	O(n)	Yes
Hash table	O(1)	O(n)	O(1)	O(n)	O(1)	O(n)	No
Self-balancing binary search tree	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)	Yes
Unbalanced binary search tree	O(log n)	O(n)	O(log n)	O(n)	O(log n)	O(n)	Yes

The last 3 will be covered later.

Dictionary implementations

Check out the code:
salist, saldict