**ADVANCED DATA STRUCTURES**

A data structure is a way of storing data that implements certain operations. When choosing a data structure for your ADT, you might consider many issues such as whether the data is static or dynamic, whether the deletion operation is important, and whether the data is ordered. In general “A data structure should take ownership of its data”.

1. **Bag**

A bag is an unordered collection of objects, where you can keep adding objects to the bag. So with a bag data structure, you can collect all the objects, and then iterate through them. You will bags normally when you program in Java. Bag is used to store same or different types of elements in it. The elements are not ordered, they are in random order i.e they doesn't depend on the order in which they are inserted. Sometimes a bag may contain duplicate elements. So a bag data structure should normally support operations like,

* Bag()                  -  to create an empty bag
* add(Item item)    -  to add an item to the bag
* isEmpty()            -  to check whether the bag is empty
* size()                   -  to get the number of items in the bag
* remove() - remove an item from bag

The Bag ADT might have:

• accessors methods such as size, countOccurrence, possibly an iterator (which steps through all the elements);

• modiﬁer methods such as add, remove, and addAll; and

• Also a union method which combines two bags to produce a third.

**Applications of Bags**

The bag is the most basic of collection data structures, and hence almost any application that does not require remembering the order that elements are inserted will use a variation on a bag. Take, for example, a spelling checker. An on-line checker would place a dictionary of correctly spelled words into a bag. Each word in the file is then tested against the words in the bag, and if not found it is flagged. An off-line checker could use set operations. The correctly spelled words could be placed into one bag, the words in the document placed into a second, and the difference between the two computed. Words found in the document but not the dictionary could then be printed.

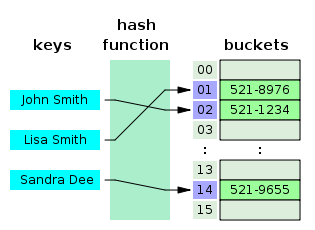
**2. Hash tables**

In [computing](https://en.wikipedia.org/wiki/Computing), a **hash table** (**hash map**) is a [data structure](https://en.wikipedia.org/wiki/Data_structure) that implements an [associative array](https://en.wikipedia.org/wiki/Associative_array) [abstract data type](https://en.wikipedia.org/wiki/Abstract_data_type), a structure that can map [keys](https://en.wikipedia.org/wiki/Unique_key) to [values](https://en.wikipedia.org/wiki/Value_(computer_science)). A hash table uses a [hash function](https://en.wikipedia.org/wiki/Hash_function) to compute an *index* into an array of *buckets* or *slots*, from which the desired value can be found.

Ideally, the hash function will assign each key to a unique bucket, but most hash table designs employ an imperfect hash function, which might cause hash [*collisions*](https://en.wikipedia.org/wiki/Collision_(computer_science)) where the hash function generates the same index for more than one key. Such collisions must be accommodated in some way.

In a well-dimensioned hash table, the average cost (number of [instructions](https://en.wikipedia.org/wiki/Instruction_(computer_science))) for each lookup is independent of the number of elements stored in the table. Many hash table designs also allow arbitrary insertions and deletions of key-value pairs, at constant average cost per operation.

In many situations, hash tables turn out to be on average more efficient than [search trees](https://en.wikipedia.org/wiki/Search_tree) or any other [table](https://en.wikipedia.org/wiki/Table_(computing)) lookup structure. For this reason, they are widely used in many kinds of computer [software](https://en.wikipedia.org/wiki/Software), particularly for associative arrays, [database indexing](https://en.wikipedia.org/wiki/Database_index), [caches](https://en.wikipedia.org/wiki/Cache_(computing)), and [sets](https://en.wikipedia.org/wiki/Set_(abstract_data_type)).

[](https://en.wikipedia.org/wiki/File:Hash_table_3_1_1_0_1_0_0_SP.svg)

A small phone book as a hash table

Operations

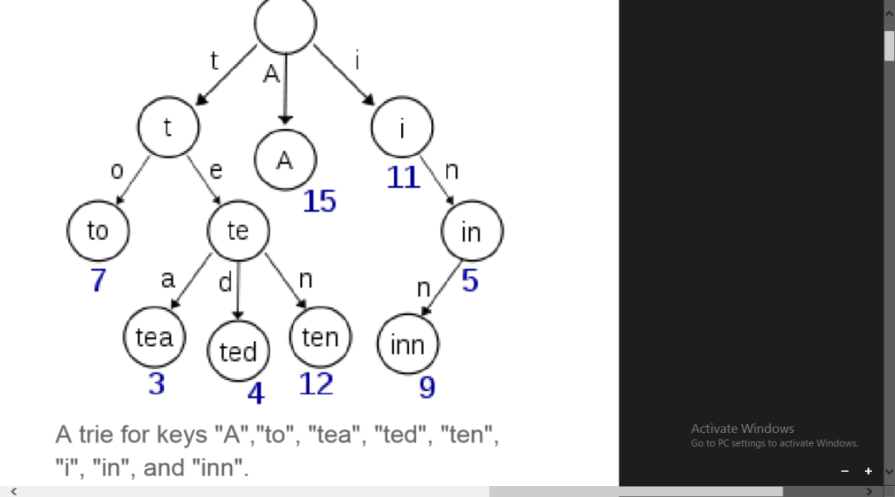
* insert -inserting new record
* delete -delete existing record
* lookup –check for particular record

**Applications**

* *Associative arrays*: Hash tables are commonly used to implement many types of in-memory tables. They are used to implement associative arrays (arrays whose indices are arbitrary strings or other complicated objects).
* *Database indexing*: Hash tables may also be used as disk-based data structures and database indices (such as in dbm).
* *Caches*: Hash tables can be used to implement caches i.e. auxiliary data tables that are used to speed up the access to data, which is primarily stored in slower media.
* *Object representation*: Several dynamic languages, such as Perl, Python, JavaScript, and Ruby use hash tables to implement objects.
* Hash Functions are used in various algorithms to make their computing faster.

**3. Trie**

In computer science, a trie, also called digital tree, radix tree or prefix tree is a kind of search tree—an ordered tree data structure used to store a dynamic set or associative array where the keys are usually strings. Unlike a binary search tree, no node in the tree stores the key associated with that node; instead, its position in the tree defines the key with which it is associated. All the descendants of a node have a common prefix of the string associated with that node, and the root is associated with the empty string. Keys tend to be associated with leaves, though some inner nodes may correspond to keys of interest. Hence, keys are not necessarily associated with every node. For the space-optimized presentation of prefix tree, see compact prefix tree.



In the example shown, keys are listed in the nodes and values below them. Each complete English word has an arbitrary integer value associated with it.

Operations on tries

* insert -inserting new node
* search - Searching
* delete

**Applications**

***As a dictionary***

Looking up if a word is in a trie takes O(n) operations, where n is the length of the word. Thus - for array implementations - the lookup speed doesn't change with increasing trie size. It has been used to store large dictionaries of English (say) words in spelling-checking programs and in natural-language "understanding" programs. Simple spell checkers operate on individual words by comparing each of them against the contents of a dictionary, possibly performing stemming on the word. If the word is not found it is considered to be an error, and an attempt may be made to suggest a word that was likely to have been intended. Word completion is straightforward to implement using a trie: simply find the node corresponding to the first few letters, and then collapse the subtree into a list of possible endings. This can be used in autocompleting user input in text editors.

***Tries and Web Search Engines***

The index of a search engine (collection of all searchable words) is stored into a compressed trie. Each leaf of the trie is associated with a word and has a list of pages (URLs) containing that word, called occurrence list. The trie is kept in internal memory. The occurrence lists are kept in external memory and are ranked by relevance. Boolean queries for sets of words (e.g. Java and coffe) correspond to set operations (e.g. intersection) on the occurrence lists. Additional information retrievel techniques are used, such as: - stopword elimination (e.g ignores “the”, “a”, “is”). - Stemming (e.g. identify “add”, “adding”, “added”). - Link analysis (recognize authoritative pages).