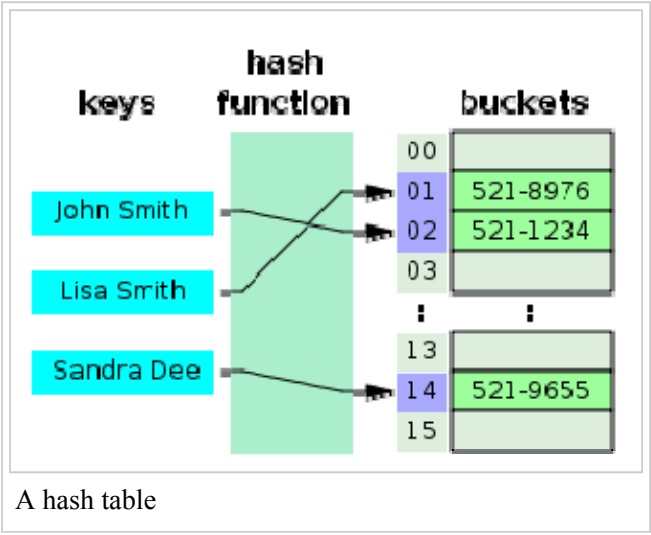
Data structure

From Wikipedia, the free encyclopedia

In computer science, a **data structure** is a particular way of storing and organizing data in a computer so that it can be used efficiently.[1][2]

Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks. For example, B-trees are particularly well-suited for implementation of databases, while compiler implementations usually use hash tables to look up identifiers.



A hash table

Data structures provide a means to manage large amounts of data efficiently, such as large databases and internet indexing services. Usually, efficient data structures are a key to designing efficient algorithms. Some formal design methods and programming languages emphasize data structures, rather than algorithms, as the key organizing factor in software design. Storing and retrieving can be carried out on data stored in both main memory and in secondary memory.

Contents

Overview

- An **array** stores a number of elements in a specific order. They are accessed using an integer to specify which element is required (although the elements may be of almost any type). Arrays may be fixed-length or expandable.
- **Records** (also called **tuples** or **structs**) are among the simplest data structures. A record is a value that contains other values, typically in fixed number and sequence and typically indexed by names. The elements of records are usually called *fields* or *members*.
- A **hash table** (also called a **dictionary** or **map**) is a more flexible variation on a record, in which name-value pairs can be added and deleted freely.

- A **union** type specifies which of a number of permitted primitive types may be stored in its instances, e.g. "float or long integer". Contrast with a record, which could be defined to contain a float *and* an integer; whereas, in a union, there is only one value at a time.
- A **tagged union** (also called a **variant**, **variant record**, **discriminated union**, or **disjoint union**) contains an additional field indicating its current type, for enhanced type safety.
- A **set** is an abstract data structure that can store specific values, without any particular order, and with no repeated values. Values themselves are not retrieved from sets, rather one tests a value for membership to obtain a boolean "in" or "not in".
- **Graphs** and **trees** are linked abstract data structures composed of *nodes*. Each node contains a value and also one or more pointers to other nodes. Graphs can be used to represent networks, while trees are generally used for sorting and searching, having their nodes arranged in some relative order based on their values.
- An **object** contains data fields, like a record, and also contains program code fragments for accessing or modifying those fields. Data structures not containing code, like those above, are called plain old data structures.

Many others are possible, but they tend to be further variations and compounds of the above.

Basic principles

Data structures are generally based on the ability of a computer to fetch and store data at any place in its memory, specified by an address—a bit string that can be itself stored in memory and manipulated by the program. Thus the record and array data structures are based on computing the addresses of data items with arithmetic operations; while the linked data structures are based on storing addresses of data items within the structure itself. Many data structures use both principles, sometimes combined in non-trivial ways (as in XOR linking).

The implementation of a data structure usually requires writing a set of procedures that create and manipulate instances of that structure. The efficiency of a data structure cannot be analyzed separately from those operations. This observation motivates the theoretical concept of an abstract data type, a data structure that is defined indirectly by the operations that may be performed on it, and the mathematical properties of those operations (including their space and time cost).

Language support

Most assembly languages and some low-level languages, such as BCPL (Basic Combined Programming Language), lack support for data structures. Many high-level programming languages and some higher-level assembly languages, such as MASM, on the other hand, have special syntax or other built-in support for certain data structures, such as vectors (one-dimensional arrays) in the C language or multi-dimensional arrays in Pascal.

Most programming languages feature some sort of library mechanism that allows data structure implementations to be reused by different programs. Modern languages usually come with standard libraries that implement the most common data structures. Examples are the C++ Standard Template Library, the Java Collections Framework, and Microsoft's .NET Framework.

Modern languages also generally support modular programming, the separation between the interface of a library module and its implementation. Some provide opaque data types that allow clients to hide implementation details. Object-oriented programming languages, such as C++, Java and Smalltalk may use classes for this purpose.

Many known data structures have concurrent versions that allow multiple computing threads to access the data structure simultaneously.

See also

- List of data structures
- Plain old data structure
- Concurrent data structure
- Data model
- Dynamization
- Linked data structure
- Persistent data structure

References

1. ^ Paul E. Black (ed.), entry for *data structure* in *Dictionary of Algorithms and Data Structures. U.S. National Institute of Standards and Technology. 15 December 2004. Online version (http://www.itl.nist.gov/div897/sqg/dads/HTML/datastructur.html) Accessed May 21, 2009.*
2. ^ Entry *data structure* in the Encyclopædia Britannica (2009) Online entry (http://www.britannica.com/EBchecked/topic/152190/data-structure) accessed on May 21, 2009.

Further reading

- Peter Brass, *Advanced Data Structures*, Cambridge University Press, 2008.
- Donald Knuth, *The Art of Computer Programming*, vol. 1. Addison-Wesley, 3rd edition, 1997.
- Dinesh Mehta and Sartaj Sahni *Handbook of Data Structures and Applications*, Chapman and Hall/CRC Press, 2007.
- Niklaus Wirth, *Algorithms and Data Structures*, Prentice Hall, 1985.
- Diane Zak, Introduction to programming with c++, copyright 2011 Cengage Learning Asia Pte Ltd

External links

- UC Berkeley video course on data structures (http://academicearth.org/courses/data-structures)
- Descriptions (http://nist.gov/dads/) from the Dictionary of Algorithms and Data Structures
- Data structures course (http://www.cs.auckland.ac.nz/software/AlgAnim/ds_ToC.html)
- An Examination of Data Structures from .NET perspective (http://msdn.microsoft.com/en-us/library/aa289148(VS.71).aspx)
- Schaffer, C. *Data Structures and Algorithm Analysis* (http://people.cs.vt.edu/~shaffer/Book/C++3e20110915.pdf)

Retrieved from "http://en.wikipedia.org/w/index.php?title=Data_structure&oldid=585319538"
Categories: Data structures