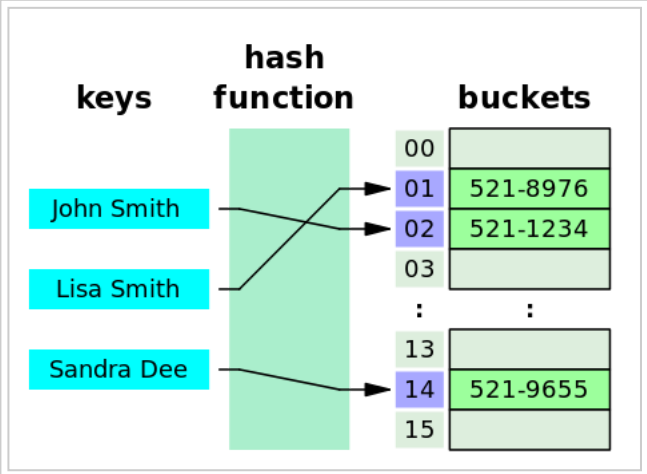
Data structure

From Wikipedia, the free encyclopedia

In computer science, a **data structure** is a particular way of organizing data in a computer so that it can be used efficiently.[1][2] Data structures can implement one or more particular abstract data types (ADT), which specify the operations that can be performed on a data structure and the computational complexity of those operations. In comparison, a data structure is a concrete implementation of the specification provided by an ADT.

Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks. For example, relational databases commonly use B-tree indexes for data retrieval,[3] 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 for uses such as large databases and internet indexing services. Usually, efficient data structures are 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. Data structures can be used to organize the storage and retrieval of information stored in both main memory and secondary memory.

Contents

Overview

Data structures are generally based on the ability of a computer to fetch and store data at any place in its memory, specified by a pointer—a bit string, representing a memory address, that can be itself stored in memory and manipulated by the program. Thus, the array and record 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).

Examples

There are numerous types of data structures, generally built upon simpler primitive data types:

- An *array* is a number of elements in a specific order, typically all of the same type. Elements are accessed using an integer index to specify which element is required (Depending on the language, individual elements may either all be forced to be the same type, or may be of almost any type). Typical implementations allocate contiguous memory words for the elements of arrays (but this is not always a necessity). Arrays may be fixed-length or resizable.
- A *linked list* (also just called *list*) is a linear collection of data elements of any type, called nodes, where each node has itself a value, and points to the next node in the linked list. The principal advantage of a linked list over an array, is that values can always be efficiently inserted and removed without relocating the rest of the list. Certain other operations, such as random access to a certain element, are however slower on lists than on arrays.
- A *record* (also called *tuple* or *struct*) is an aggregate data structure. 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 *union* is a data structure that 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. Enough space is allocated to contain the widest member datatype.
- A *tagged union* (also called *variant*, *variant record*, *discriminated union*, or *disjoint union*) contains an additional field indicating its current type, for enhanced type safety.
- A *class* is a data structure that contains data fields, like a record, as well as various methods which operate on the contents of the record. In the context of object-oriented programming, records are known as plain old data structures to distinguish them from classes.

Language support

Most assembly languages and some low-level languages, such as BCPL (Basic Combined Programming Language), lack built-in support for data structures. On the other hand, many high-level programming languages and some higher-level assembly languages, such as MASM, have special syntax or other built-in support for certain data structures, such as records and arrays. For example, the C and Pascal languages support structs and records, respectively, in addition to vectors (one-dimensional arrays) and multi-dimensional arrays.[4][5]

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

- Abstract data type
- Concurrent data structure
- Data model
- Dynamization
- Linked data structure
- List of data structures
- Persistent data structure
- Plain old 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://xlinux.nist.gov/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.
3. Gavin Powell (2006). "Chapter 8: Building Fast-Performing Database Models". *Beginning Database Design ISBN 978-0-7645-7490-0*. Wrox Publishing.
4. "The GNU C Manual". Free Software Foundation. Retrieved 15 October 2014.
5. "Free Pascal: Reference Guide". Free Pascal. Retrieved 15 October 2014.

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.

External links

- course on data structures (http://scanftree.com/Data_Structure/)
- Data structures Programs Examples in c,java (http://scanftree.com/programs/operation/data-structure/)
- UC Berkeley video course on data structures (http://academicearth.org/computer-science/)
- 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)