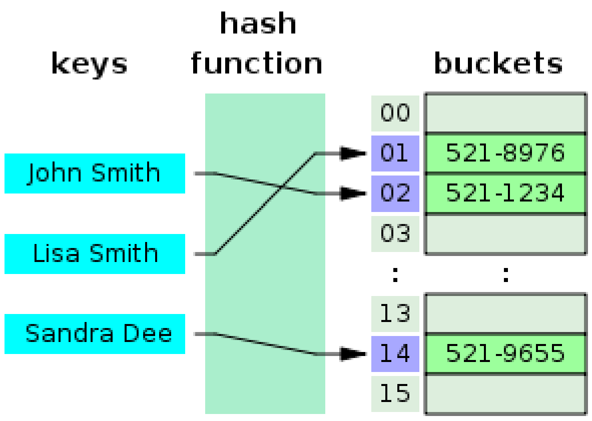
WIKIPEDIA

Data structure

In computer science, a **data structure** is a data organization, management, and storage format that enables efficient access and modification.[1][2][3] More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data.[4]



A data structure known as a hash table.

Contents

Usage

Data structures serve as the basis for abstract data types (ADT). The ADT defines the logical form of the data type. The data structure implements the physical form of the data type.[5]

Different types 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,[6] while compiler implementations usually use hash tables to look up identifiers.[7]

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.[8]

Implementation

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.

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:[9]

- An *array* is a number of elements in a specific order, typically all of the same type (depending on the language, individual elements may either all be forced to be the same type, or may be of almost any type). Elements are accessed using an integer index to specify which element is required. 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.
- An *object* is a data structure that contains data fields, like a record does, as well as various methods which operate on the data contents. An object is an in-memory instance of a class from a taxonomy. In the context of object-oriented programming, records are known as plain old data structures to distinguish them from objects.[10]

In addition, *graphs* and *binary trees* are other commonly used data structures.

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 (a direct descendant of BCPL) and Pascal languages support structs and records, respectively, in addition to vectors (one-dimensional arrays) and multi-dimensional arrays.[11][12]

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 the Microsoft .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, typically use classes for this purpose.

Many known data structures have concurrent versions which allow multiple computing threads to access a single concrete instance of a data structure simultaneously.[13]

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
- Succinct data structure

References

1. Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009). *Introduction to Algorithms, Third Edition* (https://dl.acm.org/citation.cfm?id=1614191) (3rd ed.). The MIT Press. ISBN 978-0262033848.
2. Black, Paul E. (15 December 2004). "data structure" (https://xlinux.nist.gov/dads/HTML/datastructur.html). In Pieterse, Vreda; Black, Paul E. (eds.). *Dictionary of Algorithms and Data Structures [online]*. National Institute of Standards and Technology. Retrieved 2018-11-06.
3. "Data structure" (https://www.britannica.com/technology/data-structure). *Encyclopaedia Britannica*. 17 April 2017. Retrieved 2018-11-06.
4. Wegner, Peter; Reilly, Edwin D. (2003-08-29). *Encyclopedia of Computer Science* (http://dl.acm.org/citation.cfm?id=1074100.1074312). Chichester, UK: John Wiley and Sons. pp. 507–512. ISBN 978-0470864128.
5. "Abstract Data Types" (https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/ADT.html). *Virginia Tech - CS3 Data Structures & Algorithms*.
6. Gavin Powell (2006). "Chapter 8: Building Fast-Performing Database Models" (http://searchsecurity.techtarget.com/generic/0,295582,sid87_gci1184450,00.html). *Beginning Database Design*. Wrox Publishing. ISBN 978-0-7645-7490-0.
7. "1.5 Applications of a Hash Table" (http://www.cs.uregina.ca/Links/class-info/210/Hash/). *University of Regina - CS210 Lab: Hash Table*.
8. "When data is too big to fit into the main memory" (http://homes.sice.indiana.edu/yye/lab/teaching/spring2014-C343/datatoobig.php). *homes.sice.indiana.edu*.
9. Seymour, Lipschutz (2014). *Data structures* (Revised first ed.). New Delhi, India: McGraw Hill Education. ISBN 9781259029967. OCLC 927793728 (https://www.worldcat.org/oclc/927793728).
10. Walter E. Brown (September 29, 1999). "C++ Language Note: POD Types" (https://web.archive.org/web/20161203130543/http://www.fnal.gov/docs/working-groups/fpcltf/Pkg/ISOcxx/doc/POD.html). Fermi National Accelerator Laboratory. Archived from the original (http://www.fnal.gov/docs/working-groups/fpcltf/Pkg/ISOcxx/doc/POD.html) on 2016-12-03. Retrieved 6 December 2016.
11. "The GNU C Manual" (https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html). Free Software Foundation. Retrieved 2014-10-15.
12. "Free Pascal: Reference Guide" (http://www.freepascal.org/docs-html/ref/ref.html). Free Pascal. Retrieved 2014-10-15.
13. Mark Moir and Nir Shavit. "Concurrent Data Structures" (https://www.cs.tau.ac.il/~shanir/concurrent-data-structures.pdf) (PDF). *cs.tau.ac.il*.

Bibliography

- Peter Brass, *Advanced Data Structures*, Cambridge University Press, 2008, ISBN 978-0521880374
- Donald Knuth, *The Art of Computer Programming*, vol. 1. Addison-Wesley, 3rd edition, 1997, ISBN 978-0201896831
- Dinesh Mehta and Sartaj Sahni, *Handbook of Data Structures and Applications*, Chapman and Hall/CRC Press, 2004, ISBN 1584884355
- Niklaus Wirth, *Algorithms and Data Structures*, Prentice Hall, 1985, ISBN 978-0130220059

Further reading

- Alfred Aho, John Hopcroft, and Jeffrey Ullman, *Data Structures and Algorithms*, Addison-Wesley, 1983, ISBN 0-201-00023-7
- G. H. Gonnet and R. Baeza-Yates, *Handbook of Algorithms and Data Structures - in Pascal and C (https://users.dcc.uchile.cl/~rbaeza/handbook/hbook.html)*, second edition, Addison-Wesley, 1991, ISBN 0-201-41607-7

- Ellis Horowitz and Sartaj Sahni, *Fundamentals of Data Structures in Pascal*, Computer Science Press, 1984, ISBN 0-914894-94-3

External links

- 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)