青少年信息学奥林匹克竞赛实战辅导丛书

高级数据结构（C++版）

林厚从 著

东南大学出版社
SOUTHEAST UNIVERSITY PRESS

青少年信息学奥林匹克竞赛实战辅导丛书

高级数据结构

（C++版）

林厚从 著

东南大学出版社
SOUTHEAST UNIVERSITY PRESS
·南京·

内容提要

本书在基本数据结构的基础上，围绕一些常用的高级数据结构，结合大量实战例题，深入分析“数据结构是如何服务于算法的”。本书主要内容包括：哈希表、树与二叉树、优先队列与堆、并查集、线段树、树状数组、伸展树、Treap、AVL 树、红－黑树、SBT、块状链表与块状树、后缀树与后缀数组、树链剖分与动态树等。

本书的适用对象包括：中学信息学竞赛选手及辅导老师、大学 ACM 比赛选手及教练、高等院校计算机专业的师生、程序设计爱好者等。

图书在版编目(CIP)数据

高级数据结构·C++版/林厚从著. —2 版. —南京：东南大学出版社，2017.5

ISBN 978-7-5641-4736-5

Ⅰ.①高… Ⅱ.①林… Ⅲ.①数据结构－青少年读物 Ⅳ.①TP311.12－49

中国版本图书馆 CIP 数据核字(2017)第 096699 号

高级数据结构(C++版)

责任编辑	张 煦
封面设计	余武莉
出 版 人	江建中
出版发行	东南大学出版社
社 址	南京市玄武区四牌楼 2 号(邮编 210096)
经 销	江苏省新华书店
印 刷	江苏徐州新华印刷厂
版 印 次	2017 年 5 月第 2 版 2017 年 5 月第 1 次印刷
开 本	787 mm×1092 mm 1/16
印 张	27.5
字 数	686 千字
印 数	1～3000 册
书 号	ISBN 978-7-5641-4736-5
定 价	59.00 元

(凡因印装质量问题，请直接向东大出版社市场部调换。电话：025－83791830)

丛书序

得益于计算机工具的特殊结构，以计算机技术为核心的信息技术现在已在整个社会发展中起到了极其重要的作用。同时，由于信息技术的本质在于不断创新，因而人们将21世纪称为信息世纪。根据人类生理特征，青少年时期正处于思维活跃、充满各种幻想的黄金年代，孕育着创新的种子和潜能。长期的实践活动告诉我们，青少年信息学奥林匹克竞赛可以让广大的青少年淋漓尽致地展现其思维的火花，享受创新带来的美感。因此，该项活动得到了全国各地广大青少年朋友的喜爱，越来越多的青少年朋友怀着浓厚的兴趣加入到这项活动中来。

从本质上看，计算机学科是一种思维学科，正确地思维训练可以播种持续创新的优良种子。相对于其他学科的竞赛，信息学竞赛覆盖知识面更为宽广，涉及了数学、数据结构、算法、计算几何、人工智能等相关的专业知识，如何在短时间内有效地掌握这些知识的主体，并灵活地应用其解决实际问题，显然是一个值得认真思考的问题。

知识学习与知识应用基于两种不同的思维策略，且这两种策略的统一本质上依赖于选手自身的领悟，但是如何建立两种策略之间的桥梁、快速地促进选手自身的领悟，显然是教材以及由其延伸的教学设计与实施过程所应考虑的因素。竞赛训练有别于常规的教学，要在一定的时间内得到良好的效果，需要有一定的技术方法，而不应拘泥于规范。从学习的本质看，各种显性知识的学习是相对容易的，或者说，只要时间允许，总是可以消化和理解的；然而，隐性知识的学习和掌握却是较难的。由于隐性知识的学习对竞赛和能力的提高起到决定性的作用，因此，仅仅依靠选手自身的感悟，而不从隐性知识的层面重新组织知识体系，有目的地辅助选手自身主动建构，显然是不能提高竞赛能力的。基于上述认识，结合多年来开展青少年信息学竞赛活动的经验，我们组织了一批有长期一线教学经验的教练员和专家、教授编写出版了这套《青少年信息学奥林匹克竞赛实战辅导丛书》。

丛书的主要特点如下：

1. 兼顾广大青少年课外学习时间的短暂与知识内容较多的矛盾，考虑我国青少年信息学竞赛的特点和安排，丛书分成四个层次，分别面向日常常规训练、数据结构与数学知识强化（包括基本数据结构与数学知识及应用、高级数据结构

及应用)、重点专题解析和典型试题解析,既考虑知识体系的系统性及连续训练的特点,又考虑各个层次选手独立训练的需要。

2. 区别于常规的教学模式,每册丛书的体系设计以实战需要为核心主线,突出重点,整个体系从逻辑上构成符合某种知识体系学习规律的系统化结构。

3. 围绕实战辅导需求,在解析知识和知识应用关系所蕴涵的递归思维策略的基础上,重构知识点关系,采用抛锚式和支架式并重教学思路,突出并强化知识和知识应用两者之间的联系。

4. 在显性知识及其关系基础上,强调知识应用模式及其建构的学习方法的教学,注重学习思维和能力的训练,实现知识应用能力和竞赛能力的提高,强化从程序设计及应用的角度来进行训练的特点。

5. 整套丛书的设计,不仅注重竞赛实战的需要,还考虑选手未来的发展,强调计算机程序设计正确思维的训练和培养,以不断建立持续创新的源泉。

恰逢邓小平同志"计算机的普及要从娃娃抓起"重要讲话发表25周年之际,我们期望以此奉献给广大读者朋友一套立意新、选材精、内容丰富的青少年信息学奥赛读本。

本套丛书的编写与出版,得到了东南大学出版社的大力支持,在此表示致感谢!

沈军　李立新　王晓敏

2008年12月

前　言

我们知道，计算机科学是一门研究用计算机系统进行信息表示和处理的科学。这里面涉及两个问题：信息的表示和信息的处理，而信息的表示（包括信息的组织）又直接关系到计算机程序处理信息的效率。同时，随着计算机的普及、信息量的剧增和信息范围的拓宽，许多计算机系统程序和应用程序的规模变得非常庞大，结构变得非常复杂。因此，为了编写出一个“好”的程序（或更广泛意义上所说的软件），必须深入分析待处理对象的特征及各对象之间存在的内在关系，这就是“数据结构（Data Structures）”这门学科所研究的主要问题。在大多数情况下，这些对象（包括信息及其抽象成的数据）并不是没有组织的，对象之间往往具有重要的结构关系，这就是数据结构所研究的核心内容。

一般而言，我们把堆栈、队列等使用简单、应用广泛的抽象数据结构统称为“基本数据结构”。同时，为满足一些特定需要，人们可以对一些基本数据结构和具体数据类型进行扩展，实现一些功能更为强大、具有更多操作的“高级数据结构”，如哈希表、并查集、线段树、平衡树等。严格意义上说来，数据结构并没有高级与低级之分。就具体问题而言，高级的不一定是最优的，合适的才是最好的。

本书在基本数据结构的基础上，围绕一些常用的高级数据结构，结合大量实战例题，深入分析“数据结构是如何服务于算法的”，这也是本书编写的一个核心理念：学习数据结构是为了“用好”数据结构。

本书的主要内容包括：哈希表、树与二叉树、优先队列与堆、并查集、线段树、树状数组、伸展树、Treap、AVL 树、红-黑树、SBT、块状链表与块状树、后缀树与后缀数组、树链剖分与动态树等。

本书由林厚从主编（第 1 章至第 4 章、第 7 章至第 9 章），参加编写的人员还有戴涵俊（第 5 章至第 6 章、第 10 章至第 12 章）。本书在编写的过程中，参阅和引用了 CCF NOI 的相关比赛试题以及很多 OIer 和 ACM 选手的论文、题解及相关资料，均列在本书最后的致谢中，如有遗漏请与作者联系。同时，本书的编写还得到了很多学生的帮助，他们是北京大学的吴争锴，复旦大学的陈晨、钱雨露、于竟成等。在此一并表示感谢！

应广大读者要求，2017 年 1 月，对第一版教材进行了修订，用 C++语言重新编写了所有例题的代码。在此，特别感谢北京大学的吴睿海同学。

本书的适用对象包括：中学信息学竞赛选手及辅导老师、大学 ACM 比赛选手及教练、高等院校计算机专业的师生、程序设计爱好者等。

由于计算机技术发展迅猛、技术创新不断，信息学竞赛知识范围不断扩展、难度也不断加大，而作者的知识和水平有限，书稿难免存在着一些错误或缺陷，热忱欢迎同行专家和读者朋友批评指正（联系邮箱：hc. lin@163. com），使本书在使用的过程中不断改进，日臻完善。

作　者

2017 年 5 月

知识链接

在计算机科学中，所谓“数据(Data)”就是计算机加工处理的对象。数据是信息的载体，它能够被计算机识别、存储和加工处理。数据也是计算机程序加工的原料，应用程序处理的就是各种各样的数据。数据可以是数值型数据，也可以是非数值型数据。数值型数据是一些整数、实数或复数，主要用于工程计算、科学计算和商务处理等。非数值型数据包括字符、文字、图形、图像、语音等。

“数据元素(Data Element)”是数据的基本单位。在不同的条件下，数据元素又称为元素、结点(节点)、顶点、记录等。例如，在学校的学籍管理系统中，学生信息表中的一条记录就被称为一个数据元素。有时，一个数据元素又可以由若干个数据项(Data Item)组成，例如，一条学生记录包括学生的学号、姓名、性别、籍贯、出生年月、成绩等数据项。这些数据项又可以分为两种类型：一种叫做初等项，如学生的性别、籍贯等，这些数据项是在数据处理时不能再分割的最小单位；另一种叫做组合项，如学生的成绩，它可以再划分为语文、数学、英语、物理等更小的项。通常，在解决实际问题时是把每个学生记录当做一个基本单位进行访问和处理的。

“数据对象(Data Object)”或“数据元素类(Data Element Class)”是具有相同性质的数据元素的集合。在一个具体问题中，数据元素都具有相同的性质(元素值不一定相等)，属于同一数据元素类(数据对象)，数据元素是数据元素类的一个实例。例如，在交通咨询系统的交通网中，所有的顶点是一个数据元素类，顶点 A 和顶点 B 各自代表一个城市，是该数据元素类中的两个实例，其数据元素的值分别为 A 和 B。

“数据结构”是在整个计算机科学与技术领域中广泛使用的术语。它往往用来反映一个数据的内部构成，即一个数据由哪些成分数据构成、以什么方式构成、呈什么结构。

数据结构在计算机科学界至今没有一个统一标准的定义。各人根据各自的理解的不同而有不同的表述方法。Sartaj Sahni 在他的《数据结构、算法与应用》一书中称：“数据结构是数据对象以及存在于该对象中的实例和组成实例的数据元素之间的各种联系，这些联系可以通过定义相关的函数来给出。”他将数据对象定义为“一个数据对象是实例或值的集合”。Clifford A. Shaffer 在《数据结构与算法分析》一书中有如下定义：“数据结构是抽象数据类型(Abstract Data Type，简称 ADT) 的物理实现。”Lobert L. Kruse 在《数据结构与程序设计》一书中，将一个数据结构的设计分成抽象层、数据结构层和实现层。其中，抽象层是指抽象数据类型层，它讨论数据的逻辑结构及其运算，数据结构层和实现层讨论一个数据结构的表示和在计算机内的存储细节以及运算的实现。

“数据结构”作为一门独立课程是从 1968 年开始设立的，美国的唐·欧·克努特教授开创了数据结构的最初体系，他所编著的《计算机程序设计技巧》第一卷《基本算法》是第一本较系统地阐述数据的逻辑结构和存储结构及其操作的著作。“数据结构”在计算机科学中是

一门综合性的专业基础课,是介于数学、计算机硬件和计算机软件三者之间的一门核心课程。“数据结构”这门课的内容不仅是一般程序设计(特别是非数值型程序设计)的基础,而且是设计和实现编译程序、操作系统、数据库系统及其他系统程序的重要基础。

下面,我们梳理一下数据结构中的一些基本概念和知识逻辑。

数据结构有逻辑上的数据结构和物理上的数据结构之分。逻辑上的数据结构反映成分数据之间的逻辑关系,而物理上的数据结构反映成分数据在计算机内部的存储安排,是数据结构的实现形式,是逻辑关系在计算机内的表示。此外,讨论一个数据结构必须同时讨论在该类数据上执行的运算才有意义,所以数据结构还包括数据的运算结构。

具体来说,数据是指由有限的符号组成的元素集合,结构是元素之间的关系集合。通常来说,一个数据结构 DS 可以表示为一个二元组:DS=(D,S),D 是数据元素的集合(或者是“结点”,可能含有“数据项”或“数据域”),S 是定义在 D(或其他集合)上的关系的集合,S = { R | R : D×D×... },称为元素的逻辑结构。

逻辑结构有 4 种常见的基本类型:集合结构、线性结构、树状结构和网状结构。

集合结构:该结构的数据元素之间的关系是“属于同一个集合”,别无其他关系。

线性结构:该结构的数据元素之间存在着一对一的关系。

树状结构:该结构的数据元素之间存在着一对多的关系。

网状结构:该结构的数据元素之间存在着多对多的关系,也称图状结构。树状结构和网状统称为“非线性结构”。

数据结构的物理结构是指逻辑结构的存储映射,它包括数据元素的表示和关系的表示。数据元素之间的关系有两种不同的表示(映射)方法:顺序映象和非顺序映象,并由此得到两种不同的存储结构:顺序存储结构和链式存储结构。

顺序存储结构:它是把逻辑上相邻的结点存储在物理位置相邻的存储单元里,结点间的逻辑关系由存储单元的邻接关系来体现,由此得到的存储表示方法称为顺序存储方法。顺序存储结构是一种最基本的存储表示方法,通常借助于程序设计语言中的“数组”类型来实现。

链式存储结构:它不要求逻辑上相邻的结点在物理位置上亦相邻,结点间的逻辑关系是由附加的指针字段表示的,由此得到的存储表示方法称为链式存储方法。链式存储结构通常借助于程序设计语言中的“指针”类型来实现。

一般而言,逻辑上可以把数据结构分成线性结构和非线性结构。线性结构的顺序存储结构是一种随机存取的存储结构。线性表的链式存储结构是一种顺序存取的存储结构,线性表采用链式存储表示时,所有结点之间的存储单元地址可连续也可不连续。所以,逻辑结构与数据元素本身的形式、内容、相对位置、所含结点个数都无关。

具体来说,数据结构 DS 的物理结构 P 对应于从 DS 的数据元素到存储区 M(维护着逻辑结构 S)的一个映射:P:(D,S)—>M。一个存储器 M 是一系列固定大小的存储单元,每个单元 U 有一个唯一的地址 A(U),该地址被连续地编码;每个单元 U 有一个唯一的后继单元 U′= SUCC(U)。P 有 4 种基本映射模型:顺序、链接、索引和散列。因此,根据 4 种逻辑结构,我们至少可以得到 4×4=16 种可能的物理结构。

算法的设计取决于数据的逻辑结构,而算法的实现又依赖于数据所采用的存储结构。

数据的存储结构实质上是它的逻辑结构在计算机存储器中的实现。为了全面地反映一个数据的逻辑结构，它在存储器中的映象包括两方面内容，即数据元素的信息和数据元素之间的关系。数据的运算是数据结构的一个重要方面，讨论任一种数据结构时都离不开对该结构上的数据运算及其实现算法的讨论。数据的运算是在数据的逻辑结构上定义的操作算法，如检索、插入、删除、更新和排序等。数据结构不同于数据类型，也不同于数据对象，它不仅要描述数据类型的数据对象，而且要描述数据对象各元素之间的相互关系。

“数据类型(Data Type)”是一个值的集合和定义在这个值的集合上的一组操作的总称。数据类型可分为两类：原子类型、结构类型。一方面，在程序设计语言中，每一个数据都属于某种数据类型。类型明显或隐含地规定了数据的取值范围、存储方式以及允许进行的运算。可以认为，数据类型是在程序设计中已经实现了的数据结构。另一方面，在程序设计过程中，当需要引入某种新的数据结构时，总是借助编程语言所提供的数据类型来描述数据的存储结构。计算机中表示数据的最小单位是二进制数的一位，叫做“位”。我们用一个由若干位组合起来形成的位串表示一个数据元素，通常称这个位串为“元素”或“结点”。当数据元素由若干数据项组成时，位串中对应于各个数据项的子位串称为“数据域”。元素或结点可看成数据元素在计算机中的映象。

一个软件系统框架应建立在数据之上，而不是建立在操作之上。一个含抽象数据类型的软件模块应包含定义、表示、实现三个部分。也就是说，每种数据结构类型必须要给出其特有的一些运算及其算法实现。若操作的种类和数目不同，即使逻辑结构相同，数据结构能起的作用也不同。

不同的数据结构其操作集也不同，但以下几个操作必不可缺：结构的生成、结构的销毁、在结构中查找满足给定条件的数据元素、在结构中插入新的数据元素、删除结构中已经存在的数据元素、遍历等。

“抽象数据类型(Abstract Data Type)”是指一个数学模型以及定义在该模型上的一组操作。抽象数据类型需要通过固有数据类型(高级编程语言中已实现的数据类型)来实现，是与表示无关的数据类型。对一个抽象数据类型进行定义时，必须给出它的名字及各运算的运算符名，即函数名，并且规定这些函数的参数性质。一旦定义了一个抽象数据类型及具体实现，程序设计中就可以像使用基本数据类型那样，十分方便地使用抽象数据类型。

抽象数据类型可用以下三元组表示：(D,S,P)，D 是数据对象，S 是 D 上的关系集，P 是对 D 的基本操作集。

ADT 的定义如下：

```
ADT 抽象数据类型名{
    数据对象：(数据元素集合)
    数据关系：(数据关系二元组集合)
    基本操作：(操作函数的罗列)
} ADT 抽象数据类型名；
```

抽象数据类型有两个重要特性：数据抽象(用 ADT 描述程序处理的实体时，强调的是其本质的特征、其所能完成的功能以及它和外部用户的接口，即外界使用它的方法)和数据封装(将实体的外部特性和其内部实现细节分离，并且对外部用户隐藏其内部实现细节)。

对于相同的算法,采用不同的数据结构表示其中的抽象数据类型也会造成不同的执行效率。所以,有必要研究各种抽象数据类型用不同的数据结构表示时的效率差异及其适用场合。在许多类型的程序设计中,数据结构的选择是一个基本的设计考虑因素。许多大型系统的构造经验表明,系统实现的困难程度和系统构造的质量都严重依赖于是否选择了最优的数据结构。许多时候,确定了数据结构后,算法就容易得到了。有些时候事情也会反过来,我们根据特定算法来选择数据结构与之适应。不论哪种情况,选择合适的数据结构都是非常重要的。"是'数据'而不是'算法"才是系统构造的关键因素",这种洞见导致了许多种软件设计方法和程序设计语言的出现,面向对象的程序设计语言就是其中之一。

对于抽象数据类型,我们常用的有三种基本结构:线性结构、树状结构、网状结构。线性结构是由有限个数据元素组成的有序集合,这种数据结构具有均匀性和有序性(除了首尾元素外,每个元素具有唯一的前趋和后继),包括两种不同存储结构的线性表:数组(顺序存储结构)和链表(链式存储结构,包括单链表、双链表和循环链表)。当然,也包括加了某些限制条件的、不同存取方式的特殊线性表:栈(先进后出)和队列(先进先出)。另外,还包括稍微复杂一些的数据结构:哈希表、优先队列和后缀数组等。线性结构是一种最简单、最基础的数据结构。但是,世界万物之间的联系并非都是"一对一"的线性关系,更多的是"一对多"的树状结构和"多对多"的网状结构。树状结构是一个具有层次结构的集合,除了根结点外,每个元素具有唯一的前趋;除了叶结点外,每个元素具有多个后继。树状结构可以通过先根遍历或后根遍历等方式转化为线性结构。树状结构中应用最广泛的是二叉树,任何有序树都可以转化为对应的二叉树。二叉树不仅结构简单,节省存储空间,而且基于其有序的特点,特别容易进行二分处理。在二叉树的基础上还发展出了很多更重要的数据结构,如二叉排序树、哈夫曼二叉树、字典树、堆、线段树、树状数组、伸展树、Treap 以及 AVL 树、红-黑树、SBT 等平衡树。树状结构在数据处理中发挥着重要作用,它的效率一般要好于线性结构,但有时也会退化成线性结构。网状结构又称为图结构,是表示不同事物间千变万化、错综复杂关系的数学模型。严格来说,线性结构和树状结构都是特殊的网状结构。通过图的遍历(深度优先遍历或广度优先遍历)可以将这种结构转化为线性结构。图结构可以用多种存储方式实现,如邻接矩阵、邻接表、边集数组等,不同存储结构在不同算法实现时的编程复杂度和效率上具有很大的差异,实际应用时要根据具体问题和操作频率来选择。应用图结构最困难、最具挑战性的一点就是如何建立一个抽象、高效、恰当的图论模型,常见的图论模型有最小生成树、二分图等。

目　录

第1章 哈希表

如果要存储和使用线性表(1,75,324,43,1353,90,46),那么,只要定义一个一维数组A[1..7],将表中元素按先后顺序存储在数组A中。但是,这样的存储结构会给"查找算法"带来O(n)的时间开销,尤其是n很大时,效率比较差。当然,也可以采用"二分查找"提高效率。反之,为了用O(1)的时间开销实现查找,可以分析这个线性表的元素类型和范围,定义一个一维数组A[1..1353],使得A[key]=key,即线性表的key这个元素存储在A[key]中。然而这样一来,查找的时间效率高了,空间上的开销却大了,尤其是数据范围分布很广时。为了使空间开销减少,可以对这种方法进行优化,设计一个函数h(key)=key % 13,然后把key存储在A[h(hey)]中,这样一来,只要定义一个一维数组A[0..12]就足够了,这种线性表的存储结构称为"哈希表(Hash Table)"。

哈希表是一种高效的数据结构。它的最大优点就是把数据存储和查找所消耗的时间大大减少,几乎可以看成是O(1),而代价是消耗比较多的内存。在当前竞赛可利用内存空间越来越多、程序运行时间控制得越来越紧的情况下,"以空间换时间"的做法还是值得的。另外,哈希表的编程复杂度比较低也是它的优点之一。

1.1 哈希表的基本原理

哈希表的基本原理是使用一个下标范围比较大的数组A来存储元素,设计一个函数h,对于要存储的线性表的每个元素node,取一个关键字key,算出一个函数值h(key),把h(key)作为数组下标,用A[h(key)]这个数组单元来存储node。也可以简单地理解为,按照关键字为每一个元素"分类",然后将这个元素存储在相应"类"所对应的地方(这一过程称为"直接定址")。

但是,不能够保证每个元素的关键字与函数值是一一对应的,因此极有可能出现对于不同的元素却计算出了相同的函数值,这样就产生了"冲突",换句话说,就是把不同的元素分在了相同的"类"之中了。假设一个结点的关键字值为key,把它存入哈希表的过程是根据确定的函数h计算出h(key)的值,如果以该值为地址的存储空间还没有被占用,那么就把结点存入该单元;如果此值所指单元里已存储了别的结点(即发生了冲突),那么就再用另一个函数I进行映射,计算出I(h(key)),再看用这个值作为地址的单元是否已被占用了,若已被占用,则再用I映射……直至找到一个空位置将结点存入为止。当然,这只是解决"冲突"问题的一种简单方法,如何避免、减少和处理"冲突"是使用哈希表的一个难题。

在哈希表中查找元素的过程与建立哈希表的过程相似,首先计算h(key)的值,以该值为地址到基本存储区域中去查找。如果该地址对应的空间未被占用,则说明查找失败;否则

用该结点的关键字值与要找的 key 比较，如果相等则查找成功，否则要继续用函数 I 计算 I(h(key))的值……如此重复，直到求出的某地址空间未被占用(查找失败)或者比较相等(查找成功)为止。

1.2 哈希表的基本概念

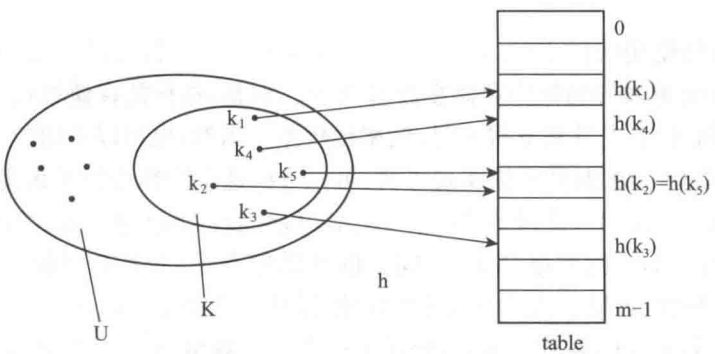

图 1-1 哈希表的基本要素示意图

图 1-1 形象地表示了哈希表的基本结构和各个要素，具体概念如下：

(1) 集合 U 是所有可能出现的关键字集合，集合 K 是实际存储的关键字集合。线性表 table 称为哈希表。

(2) 函数 h 将集合 U 映射到线性表 table[0..m−1]的下标上，可以表示成 h:U→{0,1,2,…,m−1}，通常称 h 为“哈希函数(Hash Function)”，其作用是压缩待处理的下标范围，使待处理的|U|个值减少到 m 个值，从而降低空间开销(注：|U|表示 U 中关键字的个数)。

(3) 存储地址 $h(k_i)$($k_i \in U$)称为关键字 k_i 对应结点的“哈希地址”。将结点按其关键字的哈希地址存储到哈希表中的过程称为“哈希”，这种方法称为“哈希法”。

(4) 对于关键字为 key 的结点，按照哈希函数 h 计算出地址 h(key)，若发现此地址已被别的结点占用，也就是说有两个不同的关键字值 key1 和 key2 对应到同一个地址(h(key1)=h(key2))，这个现象叫做“冲突(碰撞)”。冲突的两个(或多个)关键字称为“同义词(相对于函数 h 而言，如图中的关键字 k_2 和 k_5)”。假如先存入了 k_2，则对于 k_5，我们可以存储在$h(k_5)+1$ 中，当然 $h(k_5)+1$ 要为空，否则就逐个往后找一个空位存放，这是解决“冲突”问题的另外一种简单方法。发生了冲突就要想办法解决，必须找到另外一个新地址，这当然要降低时间效率，因此我们希望尽量减少冲突的发生。这就需要分析关键字集合的特性，找到适当的哈希函数 h 使得计算出的地址尽可能“均匀分布”在地址空间中。同时，为了提高关键字到地址转换的速度，也希望哈希函数尽量简单。对于各种取值的关键字而言，一个好的哈希函数通常只能减少冲突发生的次数，无法保证绝对不产生冲突。因此，用哈希表解题除了要选择适当的哈希函数外，还要研究发生冲突时如何解决，即用什么方法存储同义词。

(5) h(key)的值域所对应的地址空间称为“基本存储区域”，发生碰撞时，同义词可以存放在基本存储区域还没有被占用的单元里，也可以存放到基本存储区域以外另开辟的区域