2.1. 数据结构分类

常见的数据结构包括数组、链表、栈、队列、哈希表、树、堆、图，它们可以从“逻辑结构”和“物理结构”两个维度进行分类。

2.1.1. 逻辑结构：线性与非线性

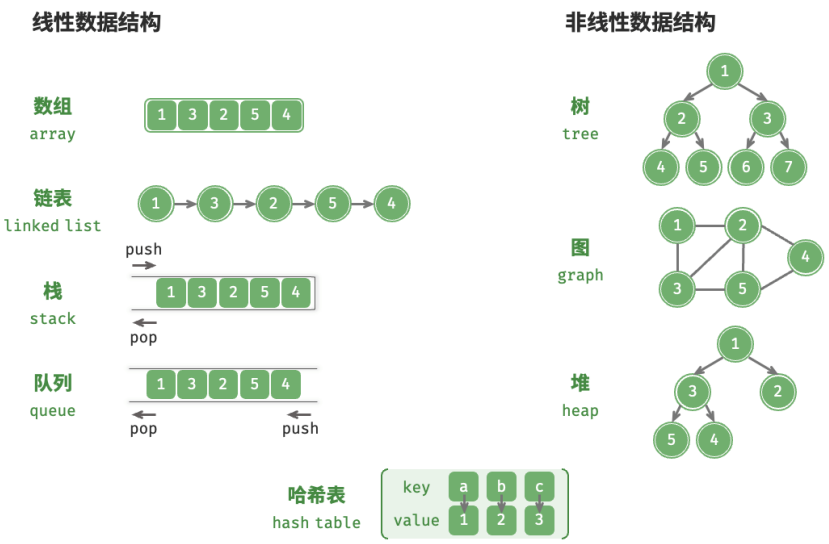
逻辑结构揭示了数据元素之间的逻辑关系。在数组和链表中，数据按照一定顺序排列，体现了数据之间的线性关系；而在树中，数据从顶部向下按层次排列，表现出“祖先”与“后代”之间的派生关系；图则由节点和边构成，反映了复杂的网络关系。

如下图所示，逻辑结构可分为“线性”和“非线性”两大类。线性结构比较直观，指数据在逻辑关系上呈线性排列；非线性结构则相反，呈非线性排列。

- **线性数据结构**：数组、链表、栈、队列、哈希表，元素之间是一对一的顺序关系。
- **非线性数据结构**：树、堆、图、哈希表。

非线性数据结构可以进一步划分为树形结构和网状结构。

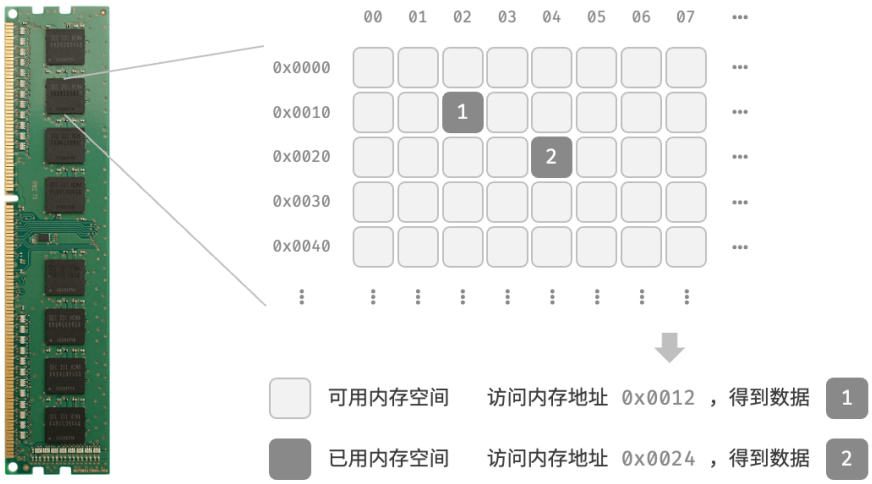
- **树形结构**：树、堆、哈希表，元素之间是一对多的关系。
- **网状结构**：图，元素之间是多对多的关系。



2.1.2. 物理结构：连续与分散

当算法程序运行时，正在处理的数据主要存储在内存中。下图展示了一个计算机内存条，其中每个黑色方块都包含一块内存空间。我们可以将内存想象成一个巨大的Excel表格，其中每个单元格都可以存储一定大小的数据。

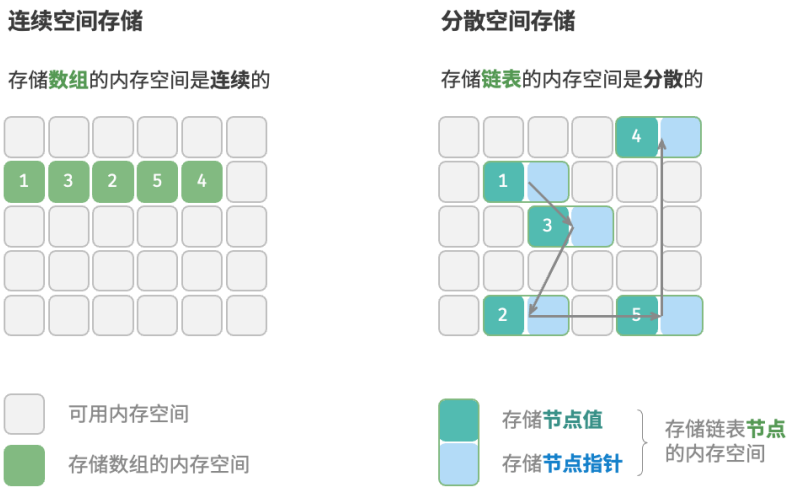
系统通过内存地址来访问目标位置的数据。如下图所示，计算机根据特定规则为表格中的每个单元格分配编号，确保每个内存空间都有唯一的内存地址。有了这些地址，程序便可以访问内存中的数据。



> **Note**
>
> 值得说明的是，将内存比作Excel表格是一个简化的类比，实际内存的工作机制比较复杂，涉及地址空间、内存管理、缓存机制、虚拟内存和物理内存等概念。

内存是所有程序的共享资源，当某块内存被某个程序占用时，则无法被其他程序同时使用了。**因此在数据结构与算法的设计中，内存资源是一个重要的考虑因素**。比如，算法所占用的内存峰值不应超过系统剩余空闲内存；如果缺少连续大块的内存空间，那么所选用的数据结构必须能够存储在分散的内存空间内。

如下图所示，**物理结构反映了数据在计算机内存中的存储方式**，可分为连续空间存储（数组）和分散空间存储（链表）。物理结构从底层决定了数据的访问、更新、增删等操作方法，两种物理结构在时间效率和空间效率方面呈现出互补的特点。



值得说明的是，**所有数据结构都是基于数组、链表或二者的组合实现的**。例如，栈和队列既可以使用数组实现，也可以使用链表实现；而哈希表的实现可能同时包含数组和链表。

- **基于数组可实现**：栈、队列、哈希表、树、堆、图、矩阵、张量（维度 ≥ 3 的数组）等。

- **基于链表可实现**：栈、队列、哈希表、树、堆、图等。

链表在初始化后，仍可以在程序运行过程中对其长度进行调整，因此也称“动态数据结构”。数组在初始化后长度不可变，因此也称“静态数据结构”。值得注意的是，数组可通过重新分配内存实现长度变化，从而具备一定的“动态性”。

2.2. 基本数据类型

当谈及计算机中的数据时，我们会想到文本、图片、视频、语音、3D模型等各种形式。尽管这些数据的组织形式各异，但它们都由各种基本数据类型构成。

基本数据类型是CPU可以直接进行运算的类型，在算法中直接被使用，主要包括以下几种。

- 整数类型 `byte` 、 `short` 、 `int` 、 `long` 。
- 浮点数类型 `float` 、 `double` ，用于表示小数。
- 字符类型 `char` ，用于表示各种语言的字母、标点符号甚至表情符号等。
- 布尔类型 `bool` ，用于表示“是”与“否”判断。

基本数据类型以二进制的形式存储在计算机中。一个二进制位即为 1 比特。在绝大多数现代操作系统中， 1 字节（byte）由 8 比特（bit）组成。

基本数据类型的取值范围取决于其占用的空间大小。下面以Java为例。

- 整数类型 `byte` 占用 1 字节 $= 8$ 比特 ，可以表示 2^8 个数字。
- 整数类型 `int` 占用 4 字节 $= 32$ 比特 ，可以表示 2^{32} 个数字。

下表列举了Java中各种基本数据类型的占用空间、取值范围和默认值。此表格无须死记硬背，大致理解即可，需要时可以通过查表来回忆。

类型	符号	占用空间	最小值	最大值	默认值
整数	`byte`	1 字节	$-2^7 (-128)$	$2^7 - 1 (127)$	0
	`short`	2 字节	-2^{15}	$2^{15} - 1$	0
	`int`	4 字节	-2^{31}	$2^{31} - 1$	0
	`long`	8 字节	-2^{63}	$2^{63} - 1$	0
浮点数	`float`	4 字节	1.175×10^{-38}	3.403×10^{38}	$0.0f$
	`double`	8 字节	2.225×10^{-308}	1.798×10^{308}	0
字符	`char`	2 字节	0	$2^{16} - 1$	0
布尔	`bool`	1 字节	false	true	false

请注意，上表针对的是Java的基本数据类型的情况。每种编程语言都有各自的数据类型定义，它们的占用空间、取值范围和默认值可能会有所不同。

- 在Python中，整数类型 `int` 可以是任意大小，只受限于可用内存；浮点数 `float` 是双精度 64 位；没有 `char` 类型，单个字符实际上是长度为 1 的字符串 `str` 。
- C和C++未明确规定基本数据类型的大小，而因实现和平台各异。上表遵循LP64数据模型，其用于包括Linux和macOS在内的Unix 64位操作系统。
- 字符 `char` 的大小在C和C++中为 1 字节，在大多数编程语言中取决于特定的字符编码方法，详见“字符编码”章节。
- 即使表示布尔量仅需 1 位（ `0` 或 `1` ），它在内存中通常也存储为 1 字节。这是因为现代计算机CPU通常将 1 字节作为最小寻址内存单元。

那么，基本数据类型与数据结构之间有什么联系呢？我们知道，数据结构是在计算机中组织与存储数据的方式。这句话的主语是“结构”而非“数据”。

如果想表示“一排数字”，我们自然会想到使用数组。这是因为数组的线性结构可以表示数字的相邻关系和顺序关系，但至于存储的内容是整数 `int` 、小数 `float` 还是字符 `char` ，则与“数据结构”无关。

换句话说，**基本数据类型提供了数据的“内容类型”，而数据结构提供了数据的“组织方式”**。例如以下代码，我们用相同的数据结构（数组）来存储与表示不同的基本数据类型，包括 `int` 、 `float` 、 `char` 、 `bool` 等。

```
# 使用多种基本数据类型来初始化数组
numbers: list[int] = [0] * 5
decimals: list[float] = [0.0] * 5
# Python 的字符实际上是长度为 1 的字符串
characters: list[str] = ['0'] * 5
bools: list[bool] = [False] * 5
# Python 的列表可以自由存储各种基本数据类型和对象引用
data = [0, 0.0, 'a', False, ListNode(0)]
```

2.3. 小结

- 数据结构可以从逻辑结构和物理结构两个角度进行分类。逻辑结构描述了数据元素之间的逻辑关系，而物理结构描述了数据在计算机内存中的存储方式。
- 常见的逻辑结构包括线性、树状和网状等。通常我们根据逻辑结构将数据结构分为线性（数组、链表、栈、队列）和非线性（树、图、堆）两种。哈希表的实现可能同时包含线性数据结构和非线性数据结构。
- 当程序运行时，数据被存储在计算机内存中。每个内存空间都拥有对应的内存地址，程序通过这些内存地址访问数据。
- 物理结构主要分为连续空间存储（数组）和分散空间存储（链表）。所有数据结构都是由数组、链表或两者的组合实现的。
- 计算机中的基本数据类型包括整数 `byte` 、 `short` 、 `int` 、 `long` ，浮点数 `float` 、 `double` ，字符 `char` 和布尔 `bool` 。它们的取值范围取决于占用空间大小和表示方式。
- 原码、反码和补码是在计算机中编码数字的三种方法，它们之间可以相互转换。整数的原码的最高位是符号位，其余位是数字的值。
- 整数在计算机中是以补码的形式存储的。在补码表示下，计算机可以对正数和负数的加法一视同仁，不需要为减法操作单独设计特殊的硬件电路，并且不存在正负零歧义的问题。
- 浮点数的编码由1位符号位、8位指数位和23位分数位构成。由于存在指数位，因此浮点数的取值范围远大于整数，代价是牺牲了精度。
- ASCII码是最早出现的英文字符集，长度为1字节，共收录127个字符。GBK字符集是常用的中文字符集，共收录两万多个汉字。Unicode致力于提供一个完整的字符集标准，收录世界上各种语言的字符，从而解决由于字符编码方法不一致而导致的乱码问题。
- UTF-8是最受欢迎的Unicode编码方法，通用性非常好。它是一种变长的编码方法，具有很好的扩展性，有效提升了存储空间的使用效率。UTF-16和UTF-32是等长的编码方法。在编码中文时，UTF-16占用的空间比UTF-8更小。Java和C#等编程语言默认使用UTF-16编码。