# 数据结构与算法

授课老师：王浩

haowang@hfut.edu.cn

合肥工業大學

HEFEI UNIVERSITY OF TECHNOLOGY

# 关于我

王浩

haowang@hfut.edu.cn

研究方向：医疗大数据与人工智能、医疗健康管理

办公地点：工程管理与智能制造大楼1404

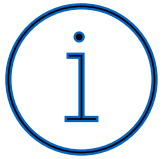
有疑问请邮件联系

# 课前的话—课程简介

- 课程时间：（1-16周）
- 教学计划教材　李春葆 等，数据结构教程（第6版），清华大学出版社，2022
- 将于课程设计一同推进理论与实践
- 校园网登录可直接阅读书籍电子版（https://lib-hfut.wqxuetang.com/deep/read/pdf?bid=3237522）

**本课程重点参考书（任选其一即可，考核内容以PPT讲义为准）：**

1. 李春葆等，数据结构教程（第6版）学习指导，清华大学出版社，2022
2. 冯舜玺 译，Mark Allen Weiss 著，数据结构与算法分析（C语言描述），机械工业出版社，2004
3. 邓俊辉，数据结构（C++语言版），清华大学出版社，2013
4. 程杰，大话数据结构（溢彩加强版），清华大学出版社，2020



**教学以C语言为载体，重点聚焦于数据结构知识本身！**

# 课前的话—规则

**数据结构与算法是一门注重实践的课程！**

**1、节奏：课程内容紧凑，不建议请假；**

**2、考核：期末40%、课堂测试30%、作业20%、综合课程报告10%；**

**3、规则：2次随堂测（8，13），开卷但不补测，作业周四课前交，不补交；**

**4、与实验课的配合：一般提前一天预告实验课内容，并对上次作业进行讲解；**

**5、实践：鼓励利用雨课堂进行拓展学习，禁止使用任何AIGC工具完成作业。**

**6、助教：每周五下午3:00-5:00 工程管理与智能制造中心1410室；**

**7、注意：无考前划重点环节，成绩相关问题请直接咨询学院教务老师；**

**8、其他：邮件 haowang@hfut.edu.cn， 一般每周五晚上8点回复。**

# 课前的话—规则

9、考试、测试及作业要求

1）写清学号、名字；

2）算法需给出**分析和注释；**

3）算法中直接使用的函数、过程先写源代码或伪代码，并说明函数功能、参数表；

4）注意算法格式(**必要的嵌套和缩进**)；

5）算法设计思路也是重要的考核依据，**不要留白；**

**6）请独立思考，禁止抄袭拷贝；**

**7）一经发现作业使用AIGC生成，直接记0分！**

# 课前的话—什么是数据结构

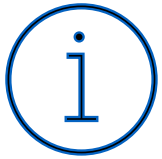
# 课前的话—了解这门课

**数据结构是一门重要的信息类专业基础课程！**

**1、基本的计算机代码能力：至少掌握一门编程语言（C）；**

**2、离散数学基础知识：了解图论相关知识；**

**3、了解基本的计算机组成原理；**



**Talk is cheap, show me the code!**

**---Linus Torvalds**

# 课前的话—课程内容

第1章：　绪论

第2章：　表

第3章：　栈和队列

第4章：　串

第5章：　数组

第6章：　树和二叉树

第7章：　图

第8章：　排序

# 第1章　绪　论

1.1　数学及C基础知识回顾

1.2　什么是数据结构

1.3　课程的基本概念

1.4　算法效率的度量

# 1.1 数学及C基础知识回顾

- **指数：**在数学中较为常用，例如 $X^A$，$2^N$ 等等，基本运算规则：
-

$$X^A X^B = X^{A+B}$$

$$\frac{X^A}{X^B} = X^{A-B}$$

$$(X^A)^B = X^{AB}$$

$$2^N + 2^N = 2^{N+1}$$

- **对数：**在表示大范围数据时，对数则比一般数要简单一些，基本规则：
-

$$\log_A B = \frac{\log_c B}{\log_c A}$$

$$\log AB = \log A + \log B$$

$$\log \frac{A}{B} = \log A - \log B$$

$$\log(A^B) = B \log A$$

# 1.1 数学及C基础知识回顾

- **级数**：一般指数据项依次累加起来的函数，例如连续级数

$$\sum_{i=0}^{N} 2^i = 2^{N+1} - 1$$

连续数之和：

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2} \approx \frac{N^2}{2}$$

# 1.1 数学及C基础知识回顾

## 1．C语言中的结构体

在C语言中表示一系列数据时可以使用数组

| 学号 | 姓名 | 性别 | 出生年 | 数学 | 英语 | 计算机原理 | 程序设计 |
|---|---|---|---|---|---|---|---|
| 100310121 | 王 刚 | 男 | 1991 | 72 | 83 | 90 | 82 |
| 100310122 | 李小明 | 男 | 1992 | 88 | 92 | 78 | 78 |
| 100310123 | 王丽红 | 女 | 1991 | 98 | 72 | 89 | 66 |
| 100310124 | 陈莉莉 | 女 | 1992 | 87 | 95 | 78 | 90 |
| … | | | | | | | |

```
long studentId[30];
char studentName[30][10];
int yearOfBirth[30];
int scoreMath[30];
int scoreEnglish[30];
int scoreComputer[30];
int scorePrograming[30];
```

# 1.1 数学及C基础知识回顾

## 1．C语言中的结构体

在C语言中表示一系列数据时可以使用数组

| 学号 | 姓名 | 性别 | 出生年 | 数学 | 英语 | 计算机原理 | 程序设计 |
|---|---|---|---|---|---|---|---|
| 100310121 | 王 刚 | 男 | 1991 | 72 | 83 | 90 | 82 |
| 100310122 | 李小明 | 男 | 1992 | 88 | 92 | 78 | 78 |
| 100310123 | 王丽红 | 女 | 1991 | 98 | 72 | 89 | 66 |
| 100310124 | 陈莉莉 | 女 | 1992 | 87 | 95 | 78 | 90 |
| … | | | | | | | |

```
long studentId[30] = {100310121, 100310122, 100310123, 100310124};
char studentName[30][10] = {"王刚", "李小明", "王丽红", "陈莉莉"};
char studentSex[30] = {'M', 'M', 'E', 'F'};
int yearOfBirth[30] = {1991, 1992, 1991, 1992};
int scoreMath[30] = {72, 88, 98, 87};
int scoreEnglish[30] = {83, 92, 72, 95};
int scoreComputer[30] = {90, 78, 89, 78};
int scorePrograming[30] = {82, 78, 66, 90};
```

# 1.1 数学及C基础知识回顾

## 数组的解决方法

| | | | |
|---|---|---|---|
| 100310121 | 王刚 | 'M' | 1991 |
| 100310122 | 李小明 | 'M' | 1992 |
| 100310123 | 王丽红 | 'F' | 1991 |
| 100310124 | 陈莉莉 | 'F' | 1992 |
| …… | …… | …… | …… |

| | | | |
|---|---|---|---|
| 72 | 83 | 90 | 82 |
| 88 | 92 | 78 | 78 |
| 98 | 72 | 89 | 66 |
| 87 | 95 | 78 | 90 |
| …… | …… | …… | …… |

分配内存不集中，寻址效率不高
对数组赋初值时，易发生错位
结构零散，内存管理困难

# 1.1 数学及C基础知识回顾

## 结构体（struct）

把关系紧密且逻辑相关的多种不同类型的变量，组织到统一的名字之下

```
struct student {
    long studentId; //学号
    char studentName[10]; //姓名
    int yearOfBirth; //生日
    int scoreMath; //数学成绩
    int scoreEnglish; //英语成绩
    int scoreComputer; //计算机成绩
    int scorePrograming; //程序设计成绩
};
```

**声明了一个结构体类型**

**结构体的名字称为结构体标签(Structure Tag)**

**构成结构体的变量称为结构体的成员(Structure Member)**

| 学号 | 姓名 | 性别 | 出生年 | 数学 | 英语 | 计算机原理 | 程序设计 |
|---|---|---|---|---|---|---|---|
| 100310121 | 王 刚 | 男 | 1991 | 72 | 83 | 90 | 82 |
| 100310122 | 李小明 | 男 | 1992 | 88 | 92 | 78 | 78 |
| 100310123 | 王丽红 | 女 | 1991 | 98 | 72 | 89 | 66 |
| 100310124 | 陈莉莉 | 女 | 1992 | 87 | 95 | 78 | 90 |
| … | | | | | | | |

# 1.1 数学及C基础知识回顾

- **结构体变量的定义**

（1）先定义结构体类型再定义变量名

（2）在定义结构体类型的同时定义变量

（3）直接定义结构体变量（不指定结构体标签）

```
struct student {
    long studentId;
    char studentName[10];
    int yearOfBirth;
    int scoreMath;
    int scoreEnglish;
    int scoreComputer;
    int scorePrograming;
} stu1;
```

```
struct {
    long studentId;
    char studentName[10];
    int yearOfBirth;
    int scoreMath;
    int scoreEnglish;
    int scoreComputer;
    int scorePrograming;
} stu1;
```

# 1.1 数学及C基础知识回顾

- 用typedef定义数据类型名

```
typedef struct student STUDENT;
typedef struct student {
    long studentId;
    char studentName[10];
    int yearOfBirth;
    int scoreMath;
    int scoreEnglish;
    int scoreComputer;
    int scorePrograming;
}STUDENT;
```

关键字typedef为一种已存在的类型定义一个别名，并未定义新类型

struct student与STUDENT类型是同义词

```
struct  student  stu1, stu2;/*可用*/
STUDENT stu1, stu2;         /*可用!*/
student stu1, stu2;         /*不可用?*/
struct  stu1, stu2;         /*不可用?*/
```

# 1.1 数学及C基础知识回顾

**实际上结构体是对一个数据对象的一种抽象，类比Java的类**

stul:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 100310121 | 王刚 | M | 1991 | 72 | 83 | 90 | 82 |

```
STUDENT stu1 = {100012, "DS", 'M', 2001, 12, 32, 32};
```

等价于

```
struct student stu1 = {100012, "DS", 'M', 2001, 12, 32, 32};
```

## • 嵌套的结构体

- 结构体定义可嵌套——嵌套的结构体（Nested Structure）
  - 在一个结构体内包含了另一个结构体作为其成员

| 学号 | 姓名 | 性别 | 出生日期 | | | 数学 | 英语 | 计算机原理 | 程序设计 |
|---|---|---|---|---|---|---|---|---|---|
| | | | 年 | 月 | 日 | | | | |

```
STUDENT2  stu2 = {100012, "DS", 'M', {2001,1,10}, 12, 32, 32,69};
```

```
typedef struct date {
    int year;
    int month;
    int day;
} DATE;
```

```
typedef struct student2 {
    long stuID;
    char stuName[10];
    char stuSex;
    DATE birthday;
    int score[4];
} STUDENT2;
```

## • 结构体变量的引用

- 访问结构体变量的成员
  - 成员选择运算符（也称圆点运算符）

```
typedef struct student2 {
    long stuID;
    char stuName[10];
    char stuSex;
    DATE birthday;
    int score[4];
} STUDENT2;
```

```
typedef struct date {
    int year;
    int month;
    int day;
} DATE;
```

结构体变量名.成员名

```
STUDENT2 stu2;
```

```
stu2.stuID=111213;
```

- 对嵌套的结构体成员，必须以级联方式访问

```
stu2.birthday.day=12;
```

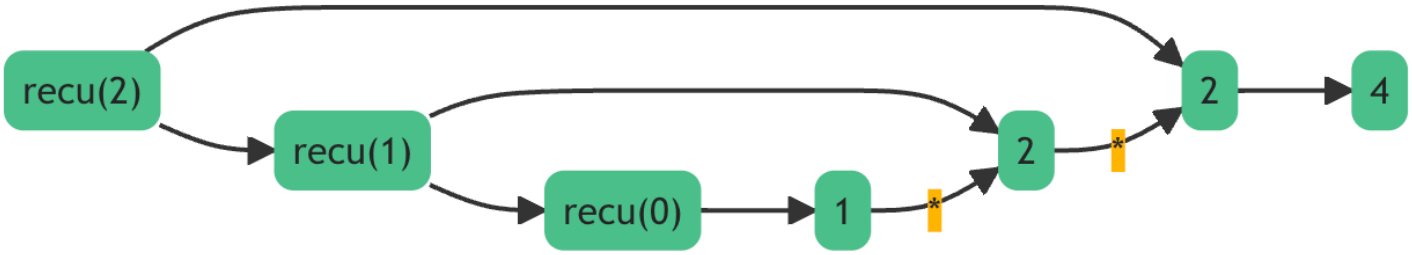
## 2. 递归

函数在定义时调用自身所形成的程序结构，称之为递归。

```c
#include <stdio.h>

int recu(int x);//函数原型

int main(void) {
    printf("Recu is %d\n", recu(2));
    return 0;
}

int recu(int x) {
    if (x == 0) {
        return 1;
    } else {
        return 2 * recu(x - 1);
    }
}
```

## 2．递归

结果为4



# 1.2 什么是数据结构与算法

## 为什么要学数据结构?

*因为要考试，所以要学这门课* 😐 。

实际上，数据结构与算法支撑着我们生活的方方面面，如购票系统、外卖点单等，可以说，任何涉及到信息系统的领域都需要数据结构的支持，并运用算法来进行问题解决。
就数据结构与算法这门课来说，不论是“信息管理与信息系统”还是“大数据”专业，《数据结构与算法》都是专业最为重要的内容， 一方面是对《程序设计基础》和《计算机导论》等课程的巩固，另一方面是为了下一阶段进一步学习《人工智能》、《计算机网络》和《数据库设计》等进阶课程的必备先修课程。

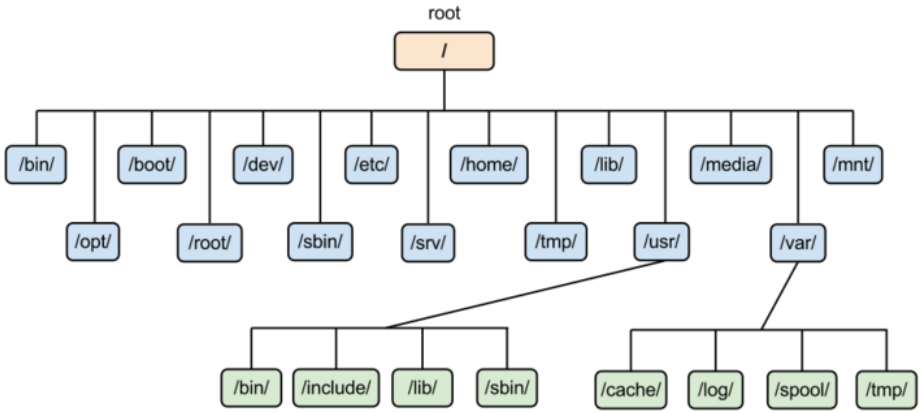
# 1.2 什么是数据结构与算法

【例1-1】图书目录表

由于表中每条记录（表示每一本书）的登录号各不相同，所以可用登录号来唯一地标识每条记录（一本图书）。在计算机的数据管理中，能唯一地标识一条记录的数据项被称为关键字。因为每本图书的登录排列位置有先后次序，所以在表中会按登录号形成一种次序关系，即整个二维表就是图书数据的一个线性序列。这种关系被称为**线性结构**。

# 1.2 什么是数据结构与算法

【例1-2】磁盘目录结构和文件管理系统




描述磁盘目录和文件结构时，假设每个磁盘包括一个根目录（root）和若干个一级子目录，每个一级子目录中又包含若干个二级子目录….

这种关系很像自然界中的树，所以称为目录树。如左图所示。

在这种结构中，目录和目录以及目录和文件之间呈现出一对多的非线性关系。即根root有多个下属（也称为后代），每一后代又有属于自己的后代；而任一个子目录或文件都只有一个唯一的上级（也称为双亲）。称这种数学模型为**树型数据结构**。

# 1.2 什么是数据结构与算法

## 【新兴案例 1】搜索

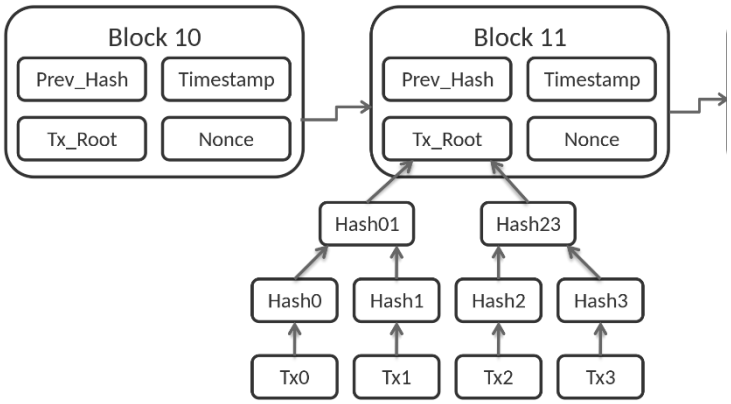
在浩瀚的信息海洋中，人们只有依靠搜索引擎才能不至于迷失方向，才能迅速找到所需的信息，也因此产生了越来越多的搜索引擎。各种搜索引擎的功能侧重并不一样，有的是综合搜索，有的是商业搜索，有的是软件搜索，有的是知识搜索。

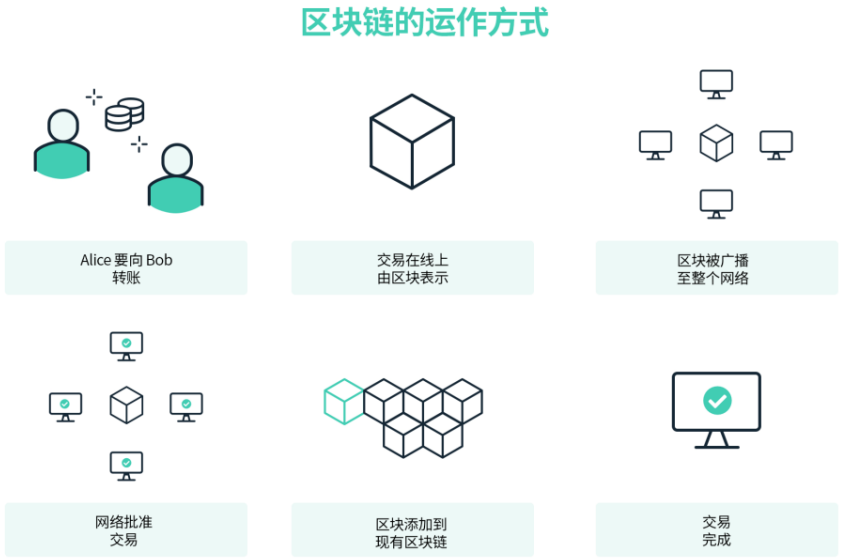
依靠单一的搜索引擎不能完全提供人们需要的信息，因此需要一种软件或网站把各种搜索引擎无缝地融合在一起，于是智能搜索引擎也随之诞生了。

# 1.2 什么是数据结构与算法

## 【新兴案例 2】区块链

区块链是一种类似于数据库的分布式账本，但不由中央机构控制，账本分布于多台计算机，这些计算机可以分布在世界各地，由任何人通过任何联网设备运行。 区块链的核心是账本，通过在这个网络中运行软件的不同节点的共识，将数据实时添加和更新到账本中。但是，数据一旦记入账本，就无法像在数据库中一样删除或编辑。



# 1.2 什么是数据结构与算法

## 从数组开始

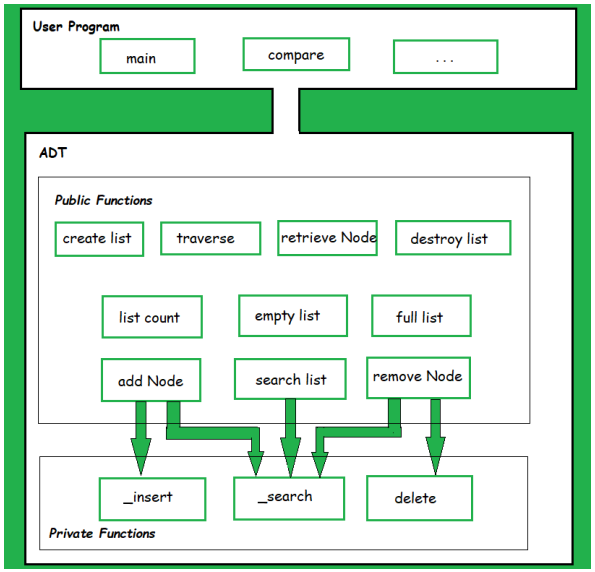
这个代码定义了一个数组，先输出了数组的第二个元素，并修改了第二个元素。Java 中还有包括更高级的Array类来实现更高级的操作。但本质上这些数据结构都包含了一些常见基本的操作，如：增加、删除、修改等。实际上反映了 ***物理上*** 数据如何存储。 为了更好地理解数据结构本身的性质，需要在逻辑上定义相应的数据结构，这里首先引入抽象数据类型（Abstract Data Type，ADT）的概念。

```c
#include <stdio.h>

int main() {
    int n[10]; /* n 是一个包含 10 个整数的数组 */
    int i, j;
    for (i = 0; i < 10; i++) { /* 初始化数组元素 */
        n[i] = i + 100; /* 设置元素 i 为 i + 100 */
    }
    for (j = 0; j < 10; j++) {   /* 输出数组中每个元素的值 */
        printf("Element[%d] = %d\n", j, n[j]);
    }
    return 0;
}
```

# 1.3 基本概念

## 1. 抽象数据类型（Abstract Data Type，简称ADT）

ADT是带有一组特定操作的对象的集合，可以视作是数学层面的抽象，它并不关注特定语言算法的实现， 而更注重相关的逻辑操作，例如增加(add)，删除(delete)，修改(modify)，查找(find)等等，常用于数据结构的表示，更多体现的是 逻辑上元素的组织及操作。在 Java 语言的设计中，也有类似 ADT 的实现，但是对其实现的细节进行了适当的隐藏。




```
public interface Collection<AnyType> extends Iterable<AnyType>
{
    int size( );
    boolean isEmpty( );
    void clear( );
    boolean contains( AnyType x );
    boolean add( AnyType x );
    boolean remove( AnyType x );
    java.util.Iterator<AnyType> iterator( );
}
```

# 1.3 基本概念

**抽象数据类型的基本要素：**

```
Class 抽象数据类型名{
    数据对象:<数据对象的定义>
    数据关系:<数据关系的定义>
    基本操作:<基本操作的定义>
}
```

**基本操作的伪码描述格式如下：**

```
基本操作名(参数表)
    初始条件:<初始条件描述>
    操作结果:<操作结果描述>
```

**可以认为ADT就是待实现的类或方法的“草稿”，主要定义了相应数据类型的行为，当然，你可以考虑使用具体的编程语言的语法来定义数据结构，如：C、C++、Java等，考虑到大家基本都学习过Java，后续我们将以Java为主进行教学，ADT仅做说明使用。**

## 2 数据结构(Data Structure)

**数据结构**是研究数据元素（Data Element）之间抽象化的相互关系（逻辑结构）和这种关系在计算机中的存储表示（物理结构），并对这种结构定义相适应的运算，设计出相应的算法。

### （1）逻辑结构：

数据之间的相互关系称为逻辑结构。通常分为 4 类基本结构：

**集合**　结构中的数据元素除了同属于一种类型外，别无其它关系。

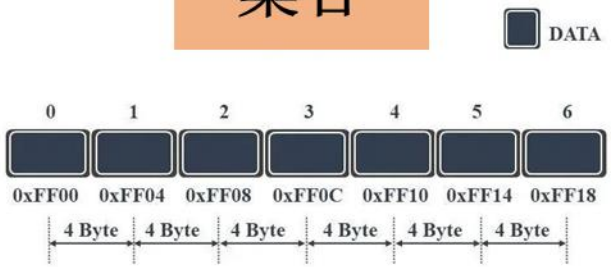
**线性结构**　结构中的数据元素之间存在一对一的关系。

**树型结构**　结构中的数据元素之间存在一对多的关系。

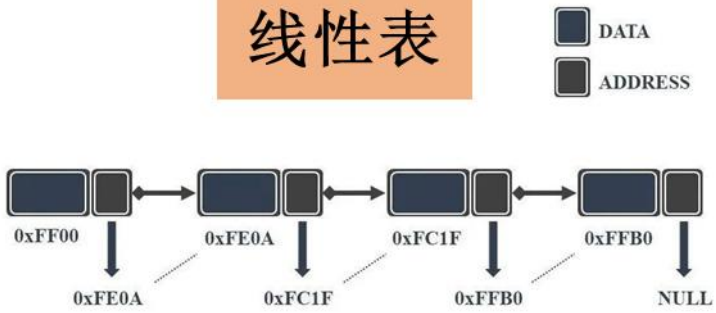
**图状结构或网状结构**　结构中的数据元素之间存在多对多的关系。

# 1.3 基本概念

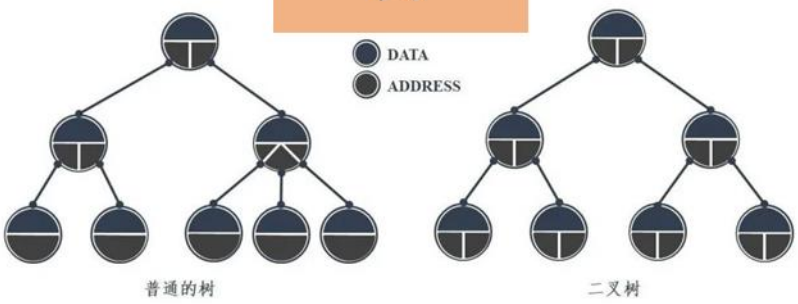
## 集合



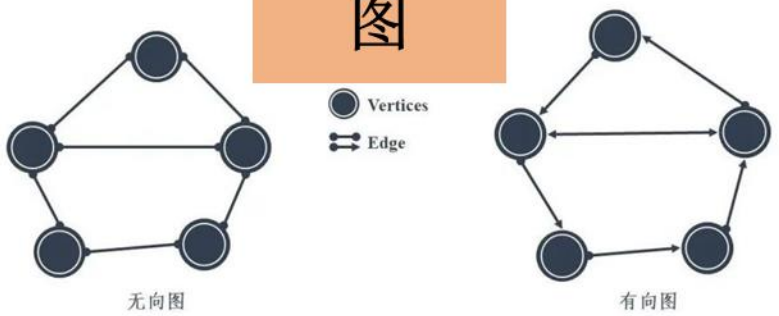

## 线性表




## 树




## 图



## （2）存储结构：

数据结构在计算机中的存储表示称为数据的存储结构。它包括数据元素的表示和关系的表示。

数据既可以存放在一块连续的内存单元中，通过元素在存储器中的位置来表示它们之间的逻辑关系（顺序），称之为**顺序**存储结构；

也可以随机分布在内存中的不同位置，通过指针元素表示数据元素之间的逻辑关系（链式），称之为**链式**存储结构。

这两种不同的表示方法的组合又可以有两种常见的存储结构：**索引**存储结构和**散列**存储结构。

# 1.3 基本概念

**顺序存储结构**是把逻辑上相邻的结点存储在物理上相邻的存储单元里，结点之间的逻辑关系由存储单元位置的邻接关系来体现。

优点：**占用较少的存储空间**；

缺点：由于只能使用相邻的一整块存储单元，因此**可能产生较多的碎片现象。**

顺序存储结构通常借助程序语言中的数组来描述。

顺序存储结构主要应用于线性的数据结构。

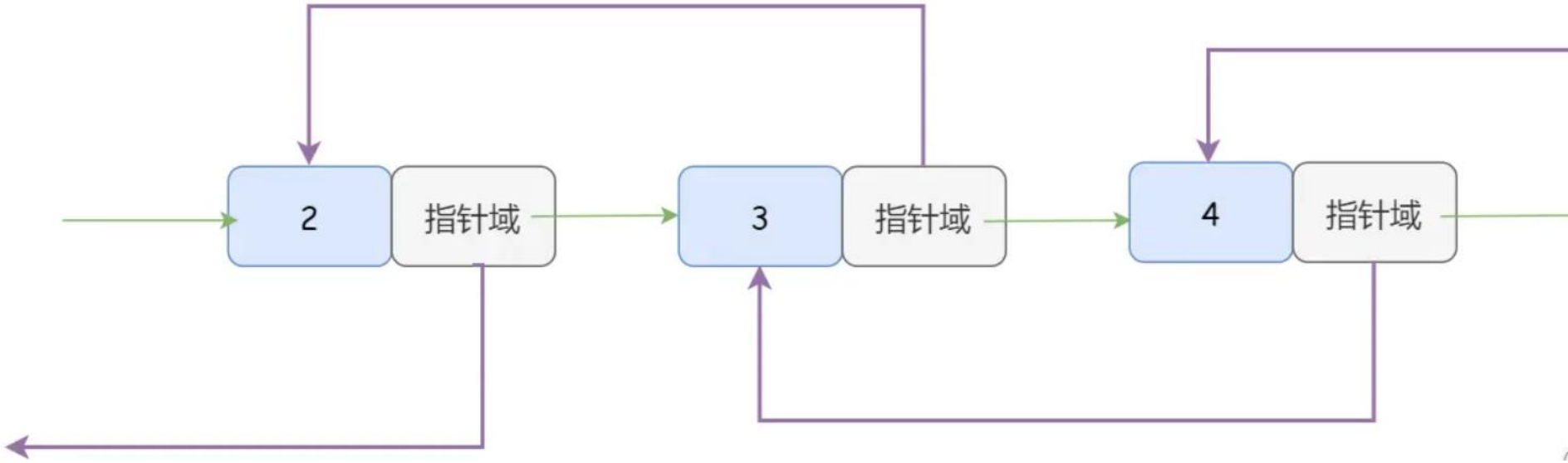
# 1.3 基本概念

**链式存储结构**将结点所占的存储单元分为两部分，一部分存放结点本身的信息，另一部分存放结点的后继结点地址，结点间的逻辑关系由附加的指针字段表示。

链式存储结构常借助于程序语言的指针类型描述。

优点：不会出现碎片现象，充分利用所有的存储单元；

缺点：每个结点占用较多的存储空间




单链表

双链表

图解数据结构：线性表 - 掘金 (juejin.cn)

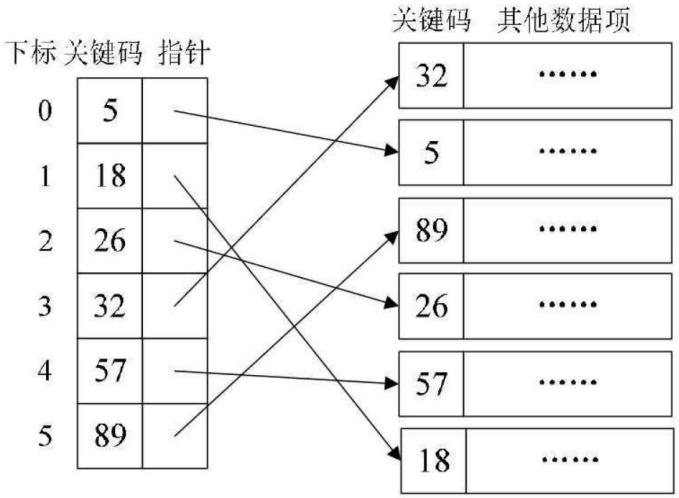
# 1.3 基本概念

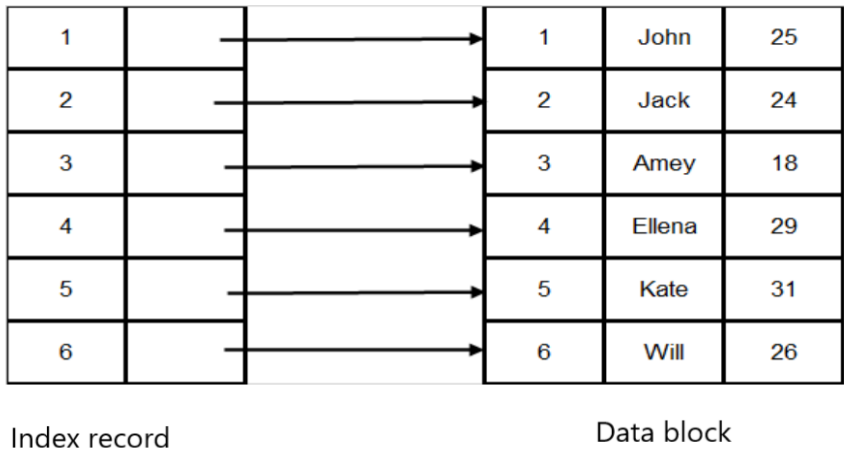
**索引存储方式是用结点的索引号来确定结点的存储地址。**

**在储存结点信息的同时，要建立附加的索引表。**

**优点：检索速度快。**

**缺点：增加了附加的索引表，占用较多的存储空间；在增加和删除数据时需要修改索引表而花费较多时间。**



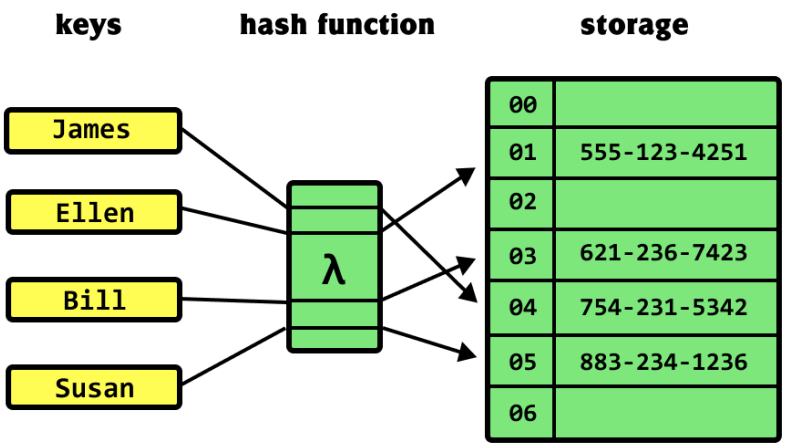



# 1.3 基本概念

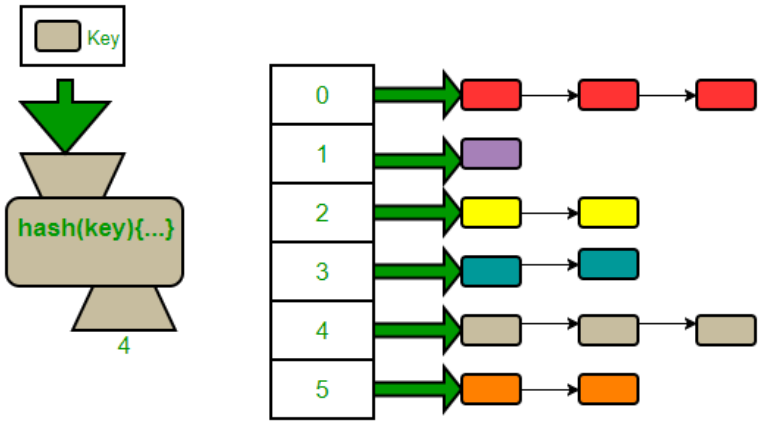
**散列存储方式**是根据结点的关键字值直接计算出该结点的存储地址。通过散列函数把结点间的逻辑关系对应到不同的物理空间。

优点：检索、增加和删除结点的操作都很快；

缺点：当采用不好的散列函数时可能出现结点存储单元的冲突，为解决冲突需要附加时间和空间的开销。




散列函数映射

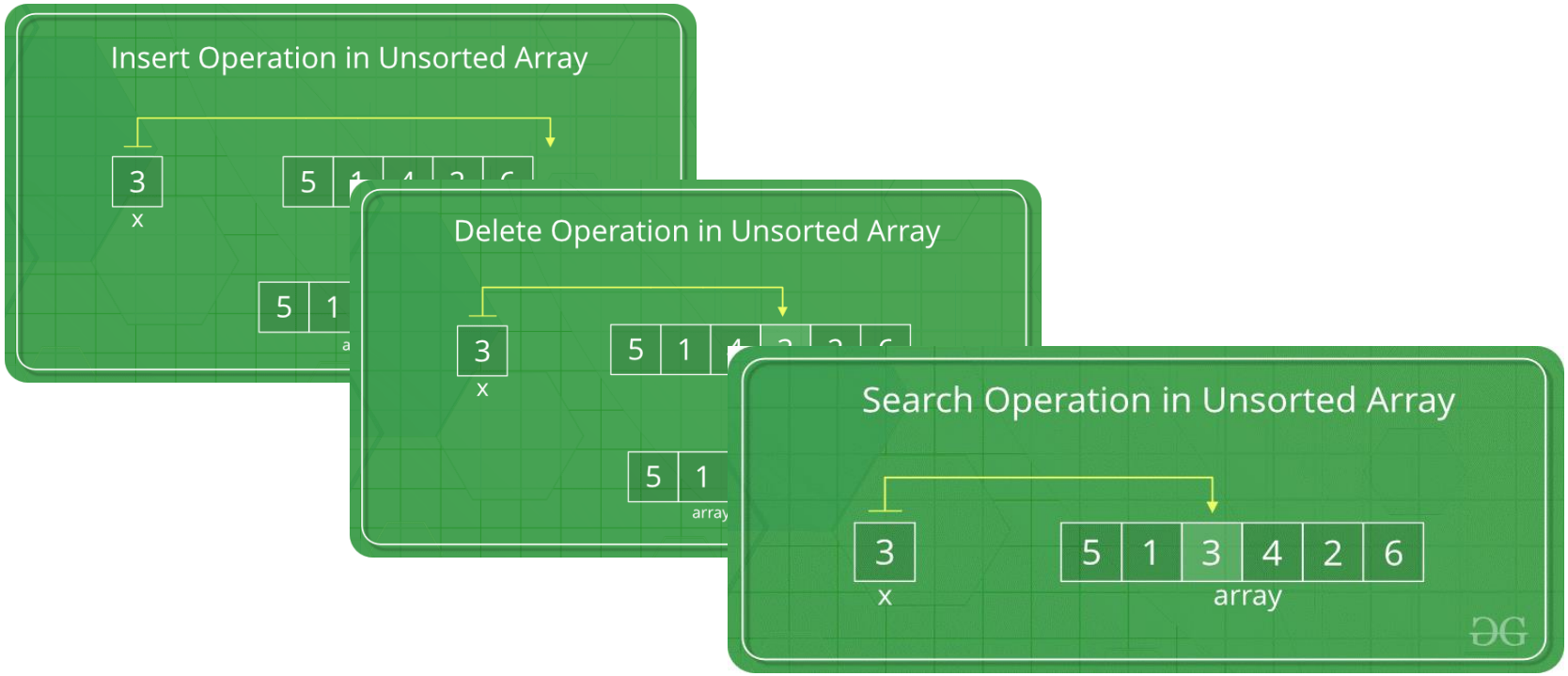



散列函数与链表

## 7 数据的运算

数据运算定义在数据的逻辑结构上，即施加于数据的操作。

例如对一张表的记录进行增加、删除、修改、查找，这就是对数据的运算。




How to search, insert, and delete in an unsorted array: - GeeksforGeeks

## 8 数据结构三方面的关系

数据的逻辑结构、数据的存储结构及数据的运算三方面构成一个数据结构的整体。

存储结构是对数据项的存储。同一逻辑结构可用不同存储结构就对应不同的存储标识。

例如，线性表若采用顺序存储方式，称为**顺序表**；若采用链式存储方式，称为**链表**；若采用散列存储方式，可称为**散列表**。

# 1.3 基本概念

**数据结构**是一门研究非数值计算的程序设计问题中计算机的操作对象以及它们之间的关系和操作的学科。数据结构与算法主要有三个方面的内容：

**数据的逻辑结构**、**数据的存储结构**和**对数据的算法**。

**逻辑结构**：反映数据之间的逻辑关系，是对数据之间关系的描述，主要有**集合**、**线性表**、**树**、**图**等结构。

**物理结构**：反映数据在计算机内部的存储安排，是数据结构在计算机中的实现方法。

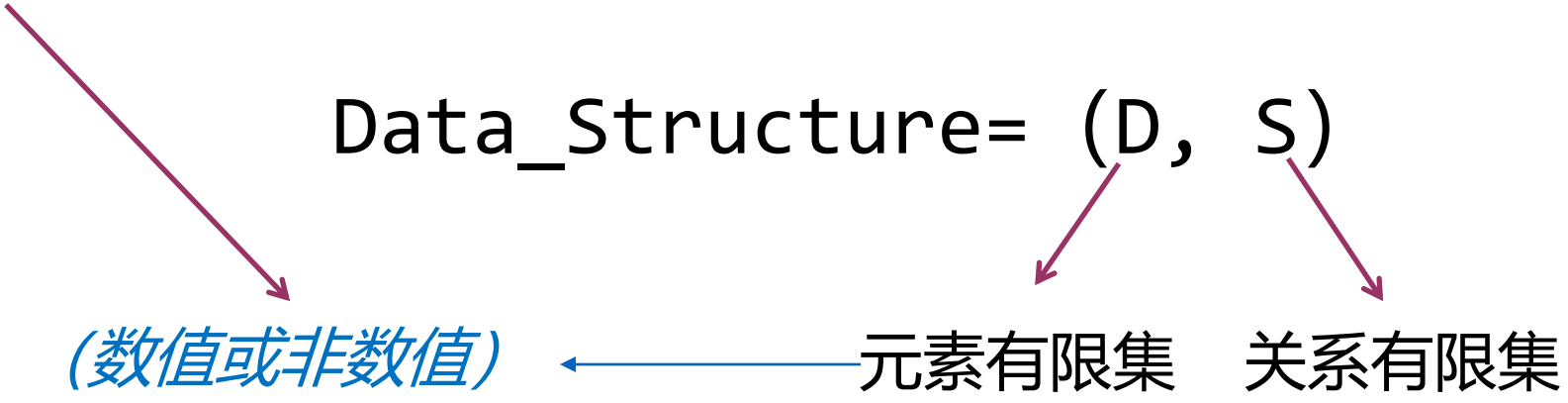
主要有**顺序**、**链接**等存储结构，并可以根据需要组合成其它更复杂的结构。

**算法**：数据进行处理的方法，与数据结构息息相关。

# 1.3 基本概念

也有人说：

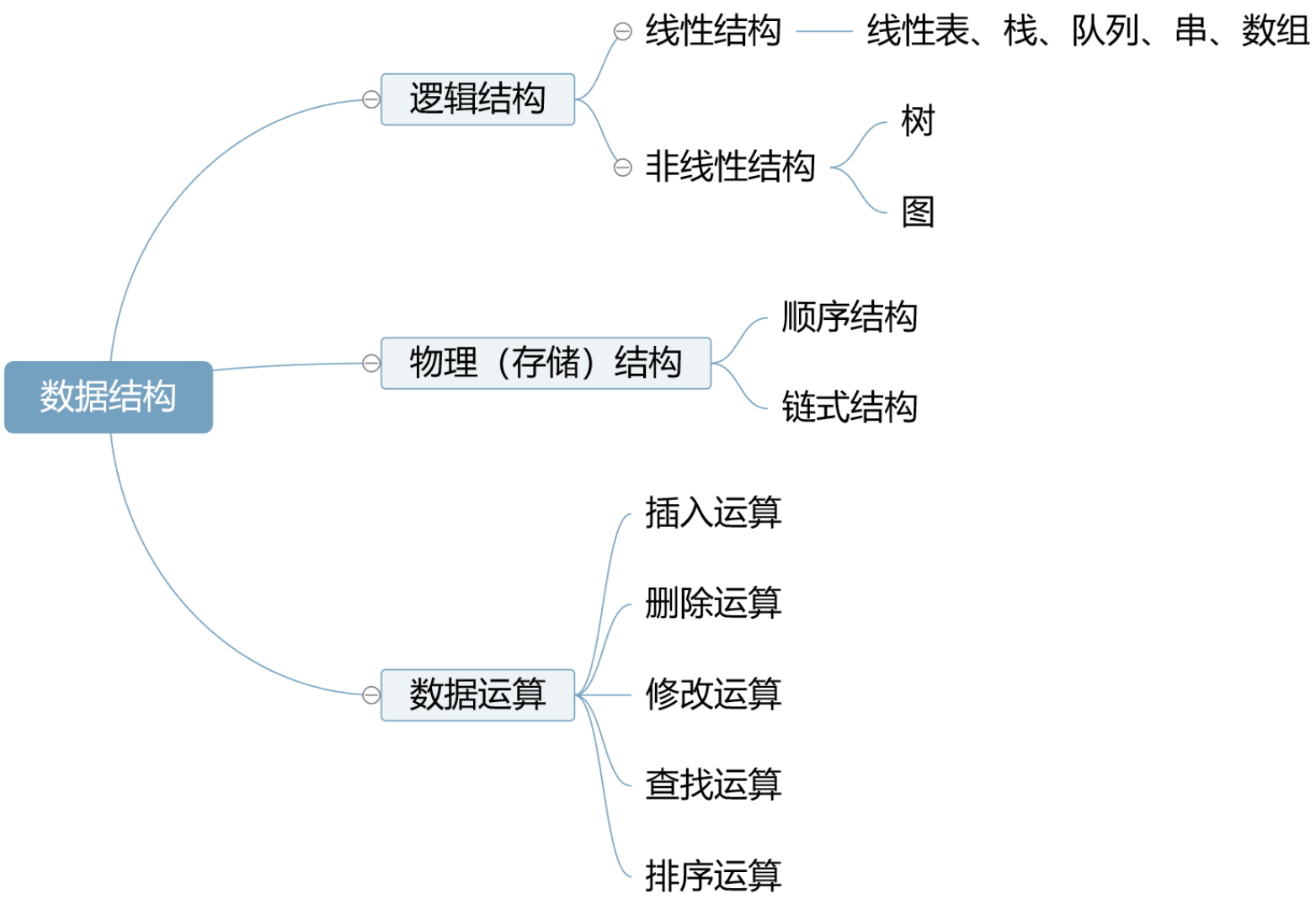
数据结构是**相互之间存在一种或多种特定关系的数据元素的集合，表示为：**




或：是指同一数据元素类中各元素之间存在的关系。

亦可表示为：S = (D, R)　　或 B= (K, R)

# 1.3 基本概念

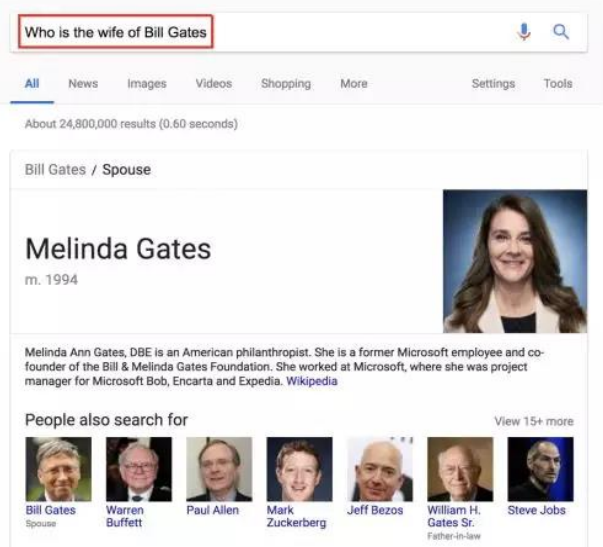


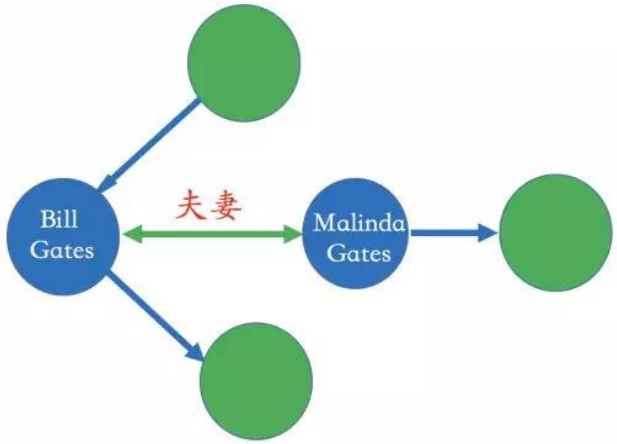

注：对于物理结构还定义顺序和链式结构组合的散列和索引结构，本课程不要求特别区分。

# 1.4 算法和算法分析

## 1.4.1 算法举例

- 搜索中的按关键字排序推荐；
- 电子商务中，按用户需求的智能匹配算法；
- 数据挖掘中的关联规则挖掘、聚类、分类等算法；



# 1.4 算法和算法分析

## 1.4.2 算法

算法（algorithm）是对特定问题求解步骤的一种描述，它是指令的有限序列，其中每一条指令表示一个或多个操作；此外算法还具有：

有穷性；确定性；可行性；输入；输出。

## 1.4.3 算法性能考量

正确性 （计算准确性）

可读性 （面向设计友好）

健壮性 （鲁棒性、容错性）

时间复杂度（程序语句执行的次数规模）

空间复杂度（申请的变量空间大小）

# 1.4 算法和算法分析

**关于算法执行时间的度量**

**（1）事后统计的方法**

**受硬件、网络的影响，实际难以准确度量算法优劣。**

**（2）事前分析估算的方法**

**影响算法耗时的因素：算法选用的计算策略、问题规模、程序语言（实现语言级别越高，执行效率越低）、编译程序锁产生的机器代码的质量、机器执行指令的速度。**

**控制结构（顺序、分支、循环）和原操作（固定数据类型操作）共同决定算法时间。**

# 1.4 算法和算法分析

随着硬件设备价格的降低以及云计算等技术的高速发展，空间复杂度在度量算法性能中已经相对弱化，但在算法设计中仍扮演着重要角色。而智能终端的广泛应用，用户对程序响应时间的考虑进一步加强，使得时间复杂度的重要性进一步加强。

在进行算法分析时，假设语句总的执行次数T(n)是关于问题规模n的函数，进而分析T(n)随n的变化情况并确定T(n)的数量级。算法的时间复杂度，也就是算法的时间量度，记作：T(n)=O(f(n))。

它表示随问题规模n的增大，算法执行时间的增长率和f(n)的增长率相同，称作算法的渐近时间复杂度，简称为时间复杂度。其中f(n)是问题规模n的某个函数。这样用大写O()来表示时间复杂度的记法，称之为大O计数法。

## 推导大O阶方法

**基本原则：**

- **循环**：for\do while\while循环的执行次数，就是其中循环语句执行的次数，记做$O(N)$;
- **循环嵌套**：注意当发生循环的嵌套时，则其复杂度为嵌套数的N次方，例如双层循环的复杂度为 $O(N^2)$（强烈建议不要进行多层循环嵌套）；
- **顺序语句**：当发生多种语句组合时，一般先统计不同组合的运行复杂度，最后取最大级数作为整体的复杂度即可；
- **if/else语句**：一般来说，条件语句合理的情况下，对于条件预计的时间复杂度计算取两个条件语句中最复杂的部分；

# 1.4 算法和算法分析

下面这段代码，它的循环的时间复杂度为O(n)，因为循环体中的代码须要执行n次。

```
/* 输出数组中每个元素的值 */
for (j = 0; j < 100; j++) {
    //时间复杂度为o(1)的语句
    printf("Element[%d] = %d\n", j, n[j]);
}
```

**下面这段代码，它的时间复杂度为O(1)，因为代码须要执行1次。**

```
int sum = 0, n = 100;    /* 执行1次 */
sum = (1 + n) * n / 2;   /* 执行第1次 */
sum = (1 + n) * n / 2; /* 执行第2次 */
sum = (1 + n) * n / 2;   /* 执行第3次 */
sum = (1 + n) * n / 2;   /* 执行第4次 */
sum = (1 + n) * n / 2;   /* 执行第5次 */
sum = (1 + n) * n / 2;   /* 执行第6次 */
sum = (1 + n) * n / 2;   /* 执行第7次 */
sum = (1 + n) * n / 2;   /* 执行第8次 */
sum = (1 + n) * n / 2;   /* 执行第9次 */
sum = (1 + n) * n / 2; /* 执行第10次 */
printf("sum is %d\n", sum);
```

# 1.4 算法和算法分析

下面的这段代码，时间复杂度又是多少呢?

```
int count = 1, n = 100;
while (count < n) {
    count = count * 2;
    /* 时间复杂度为0(1)的程序步骤序列 */
}
```

由于每次count乘以2之后，就距离n更近了一分。也就是说，有多少个2相乘后大于n，则会退出循环。由2x=n得到x=log2n。所以这个循环的时间复杂度为O(logn)。

# 1.4 算法和算法分析

**下面的这段代码，时间复杂度又是多少呢?**

```
int i, j, n = 10, m = 20;
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        printf("output is %d,%d", i, j);
        /* 时间复杂度为O(1)的程序步骤序列 */
    }
}
```

**对于外层的循环，不过是内部这个时间复杂度为O(n)的语句，再循环n次。所以这段代码的时间复杂度为$O(n^2)$，如果外循环的循环次数改为了m，时间复杂度就变为O(m×n)**

# 1.4 算法和算法分析

**下面的这段代码，时间复杂度又是多少呢?**

```
int i, j, n = 10, m = 20;
for (i = 0; i < n; i++) {
    for (j = i; j < n; j++) {
        printf("output is %d,%d", i, j);
        /* 时间复杂度为O(1)的程序步骤序列 */
    }
}
```

**由于当i=0时，内循环执行了n次，当i=1时，执行了n-1次，……当i=n-1时，执行了1次。所以总的执行次数为：O(n²)**

$$n+(n-1)+(n-2)+\cdots+1=\frac{n(n+1)}{2}=\frac{n^2}{2}+\frac{n}{2}$$

# 1.4 算法和算法分析

下面的这段代码，时间复杂度又是多少呢?

```c
#include <stdio.h>

void printNTime(int n) {
    for (int i = 0; i < n; i++) {
        printf("%d\n", i);
    }
}
int main() {
    int n = 10;                 /* 执行次数为1 */
    printNTime(n);               /* 执行次数为n */
    int i, j;
    for (i = 0; i < n; i++) { /* 执行次数为n2 */
        printNTime(i);
    }
    for (i = 0; i < n; i++) {/* 执行次数为n(n + 1)/2 */
        for (j = i; j < n; j++) {
            printf("hello\n");/* 时间复杂度为O(1)的程序步骤序列 */
        }
    }
    return 0;
}
```

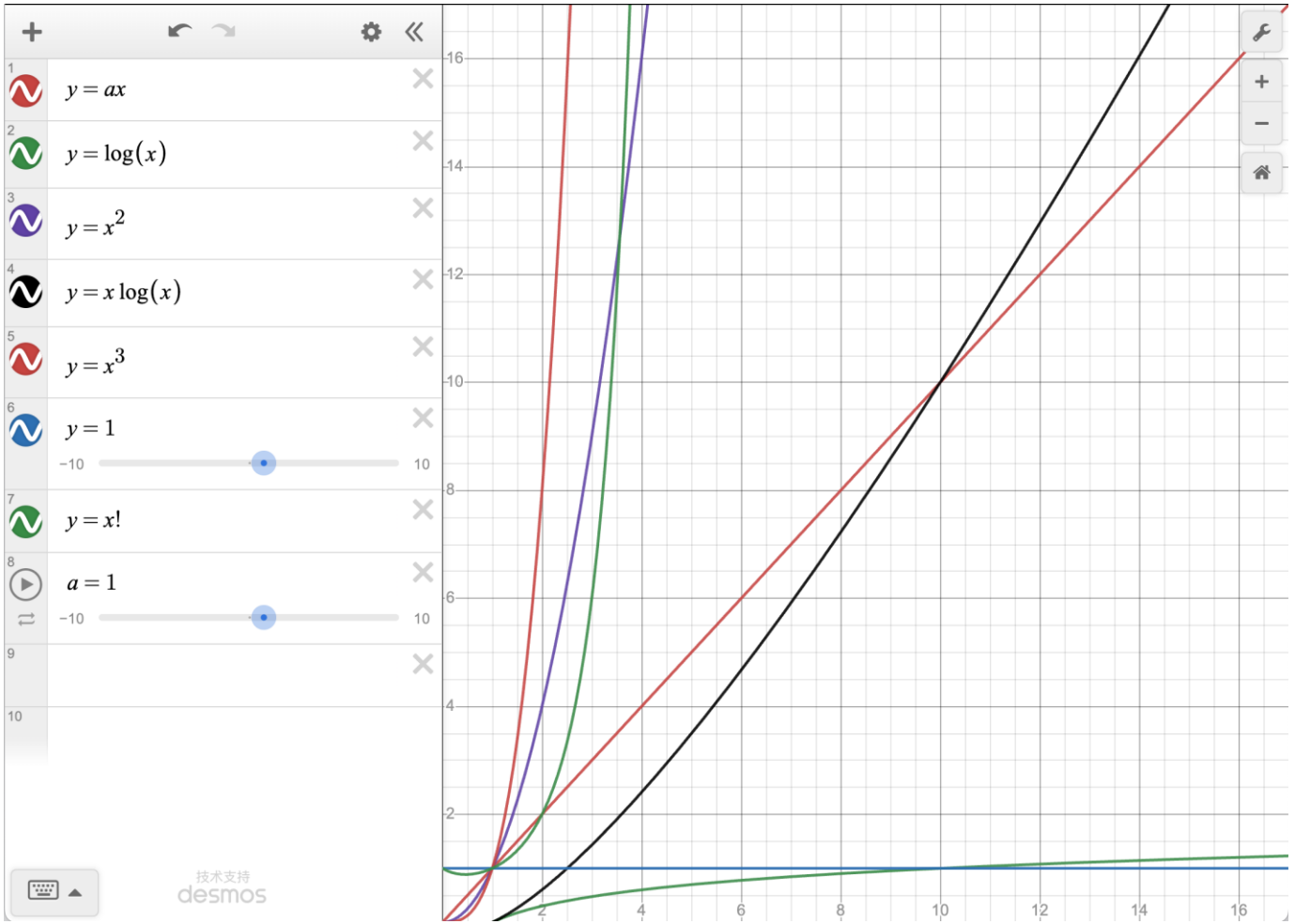
它的执行次数$f(n) = 1 + n + n^2 + \frac{n(n+1)}{2} = \frac{3}{2} \cdot n^2 + \frac{3}{2} \cdot n + 1$，根据推导大O阶的方法，最终这段代码的时间复杂度也是O($n^2$)。

# 1.4 算法和算法分析

| 执行次数 | 函数阶 | 非正式术语 |
| --- | --- | --- |
| 12 | O(1) | 常数阶 |
| 2n+3 | O(n) | 线性阶 |
| $3n^2+2n+1$ | $O(n^2)$ | 平方阶 |
| $5\log_2 n+20$ | O(logn) | 对数阶 |
| $2n+3n\log_2 n+19$ | O(nlogn) | nlogn阶 |
| $6n^3+2n^2+3n+4$ | $O(n^3)$ | 立方阶 |
| $2^n$ | $O(2^n)$ | 指数阶 |

$O(1)<O(\log n)<O(n)<O(n\log n)<O(n^2)<O(n^3)<O(2^n)<O(n!)<O(n^n)$

# 1.4 算法和算法分析


y = ax
y = log(x)
y = x^2
y = x log(x)
y = x^3
y = 1
−10
10
y = x!
a = 1
−10
10
技术支持
desmos
16
14
12
10
8
6
4
2
2
4
6
8
10
12
14
16

# 本章关键词

(1) 数据结构

(2) 存储结构

(3) 结构体

(4) 递归

(5) 算法

(6) 时间复杂度、空间复杂度

# 本章作业

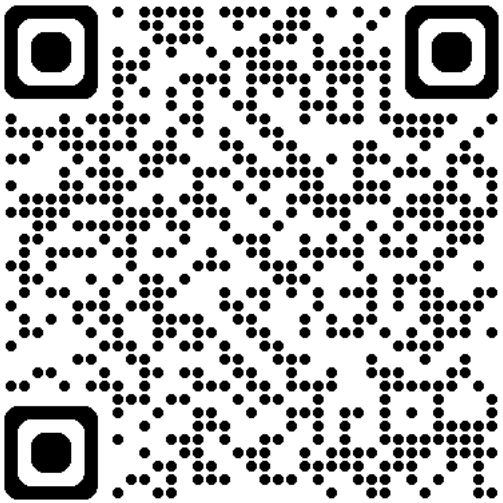
- （1）简要说明顺序存储结构、链式存储结构、索引存储结构和散列存储结构的特点，并结合自身实际举例说明四种结构的适用场景；
- （2）以递归和非递归的方式实现n的阶乘算法；
- （3）简要分析算法效率的主要度量，并以（2）为例分析影响算法时间复杂度的关键因素

作业使用word模板交即可

# 实验课预告

- （1）熟悉基本的C语言运行环境
- （2）CLion，VSCode等代码编辑器的基本配置
- （3）结构体等基本C概念的实战
- （4）递归与非递归实现n的阶乘

# Q&A



https://gitee.com/hwhfut/dsac2024