中国科学技术大学

University of Science and Technology of China

数据结构

基础知识（2 学时）

September 17, 2020

目录

编写程序时，第一个需要思考的问题

思考如下软件的编程实现：

- 图书管理系统
- 网络通信程序
- 操作系统
- 编译器
- ...

我们面对现实世界，拥有计算机内一个 0-1 构成的虚拟世界，如何连接二者？连接成功，就可以用计算机解决现实世界问题

- 基本步骤如下：
  1. 数据结构设计：将现实世界“映射”到 0-1 虚拟世界；
  2. 算法设计：考虑在 0-1 世界中的处理逻辑以及返回处理结果到现实世界。

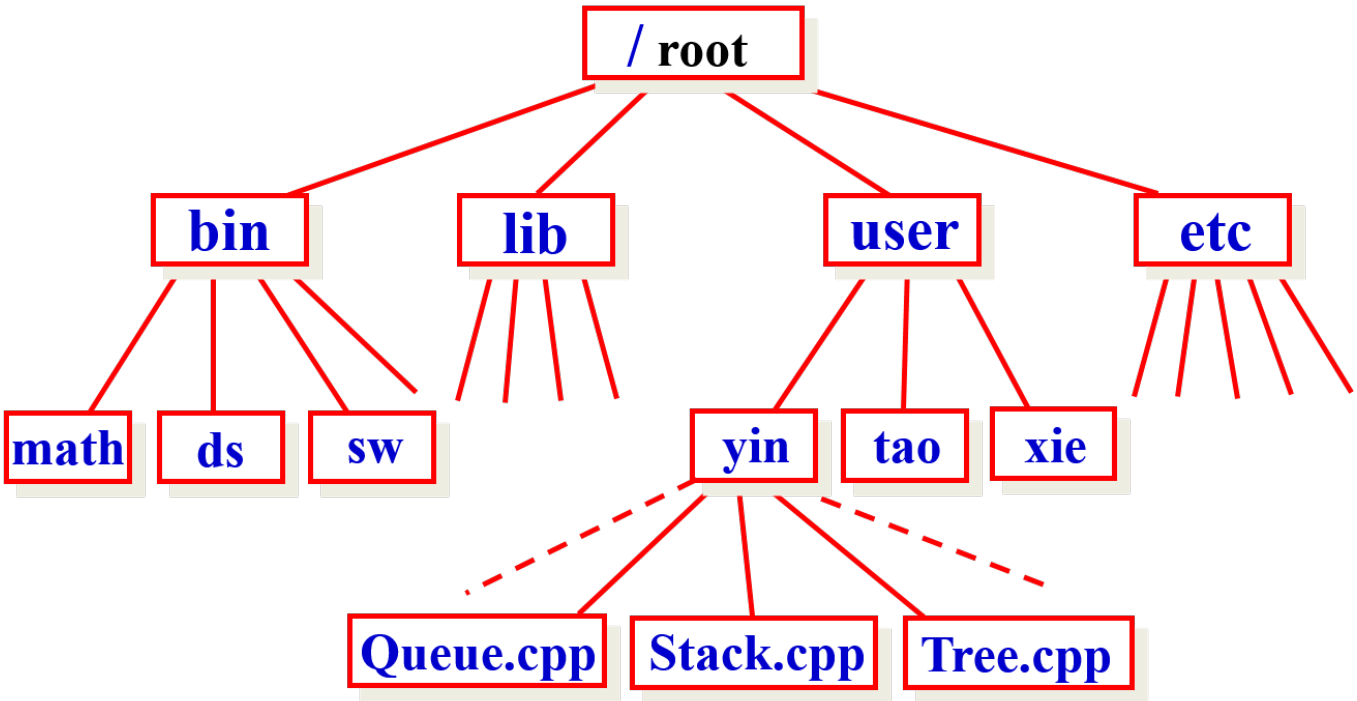
编写程序时，第一个需要思考的问题

姓名	语文	数学	英语	政治	历史	物理	生物	地理
王亦非	90	93	92	95	90	87	95	90
刘 亮	97	94	79	92	93	89	93	92
赵 鑫	92	99	86	89	91	94	94	94
李 露	96	100	97	97	93	98	89	89
陈蒙蒙	85	90	83	93	95	86	92	91
孙 洋	90	91	98	89	87	83	91	95

学生成绩表如何保存到计算机里?

- 存入内存中: 计算机内存可以想象成一排或灭或亮的灯泡，指示 0-1
- 4GBytes 的内存，就是 2^{32} 个灯泡排成一个长队，用于保存 0-1 串
- 指令和数据如何转换成 0-1 串？二维的表格如何变成一维的 0-1 串，能方便的识别和读取出来？(定长分组，8bits)

编写程序时，第一个需要思考的问题



文件系统如何实现?

- 树状的文件关系如何变成一维的 0-1 串，能方便的识别和读取其中的任何一个部件/细节?

我们将要学习的内容

- 综上，描述这类非数值计算问题的数学模型不是数学方程，而是树、表和图之类的数据结构；
- 从广义上讲，数据结构描述现实世界实体的数学模型及其上的操作在计算机中的表示和实现；
- 学习用计算机解决实际问题的关键：学会用计算机描述现实世界
  - 利用现在学会的 C 语言编程的知识：数组、链表、结构体等
  - 数据结构课程告诉你如何入手完成这些任务

课程涉及内容

- 各种常见数据结构；
- 学习算法基本概念；
- 学习与每一种数据结构相关的经典算法；
- 学习算法性能分析的基本概念和方法技巧。

数据结构与算法的关系

密不可分的数据结构和算法

- 对现实世界的同一个事物，可以设计不同的数据结构来描述它；(可能很多种)
- 事物在计算机内被描述了，接下来要处理或计算这个事物，这个处理或计算的过程（算法)，可能有多种算法；
- 不同的数据结构可能需要不同的算法来完成处理或计算；
- 数据结构的优劣常用相关算法的性能来评价。

概念与术语

数据与数据元素

- 数据 (Data)：是信息的载体，是描述客观事物的数、字符、以及所有能输入到计算机中，被计算机程序识别和处理的符号的集合；抽象概念。
  - 数值性数据
  - 非数值性数据
- 数据元素：数据的基本单位，有时一个数据元素可以由若干数据项 (Data Item) 组成。
  - 在计算机程序中常作为一个整体进行考虑和处理
  - 数据元素又称为元素、结点、记录
- 数据项：又称属性、特征，数据项是数据不可分割的最小单位。
- 大数据：属性或特征的数量非常大，一个数据即一个数据元素，数据元素包含的数据项/特征多，即数据“大”，形成“大数据”

基本概念与术语

数据对象

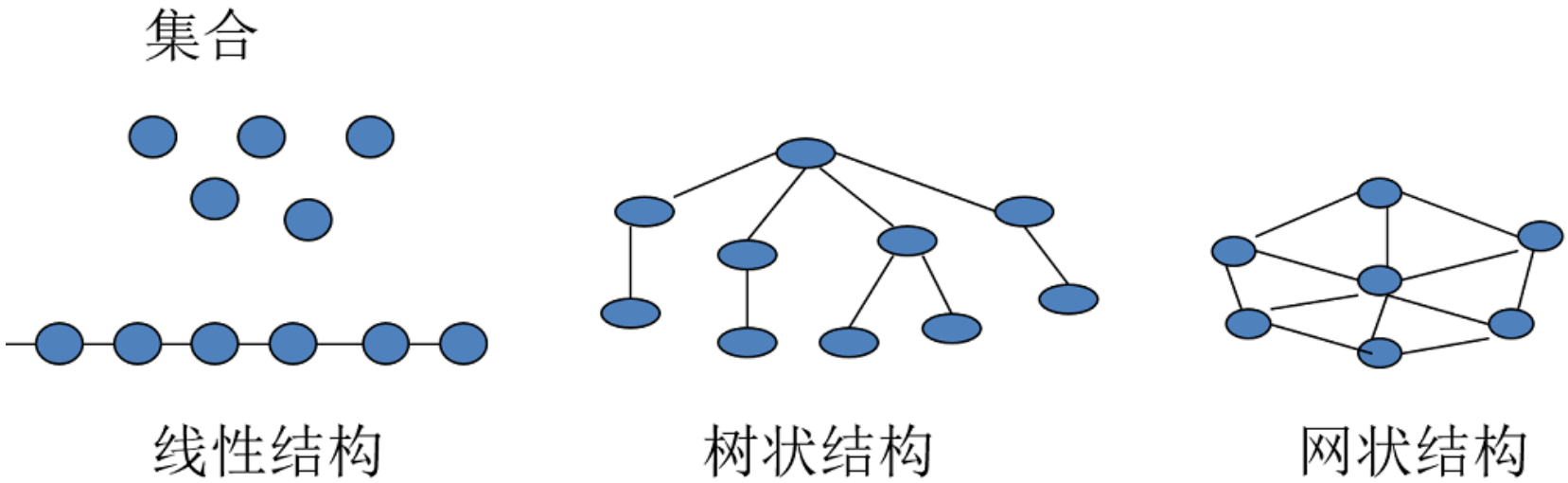
- 理解数据对象
  - 相同性质（通常指包含的属性的类型和个数一样，满足某些约束条件）的数据元素的的集合，可理解为“数据类型”
  - 比如：整数集合构成整数数据对象，26 个英文字母字符集合构成字母字符数据对象
- 数据项的集合 $\Longrightarrow$ 数据元素
- 数据元素的集合 $\Longrightarrow$ 数据对象

基本概念与术语

数据结构：某一数据对象的所有数据/元素成员之间的关系

- 记为：$DataStructure = \{D, S\}$。
  - 其中，D 是某一数据对象，S 是该对象中所有数据元素/成员之间的关系的有限集合。
- 关系：两个数据元素之间的关系
- 序偶：两个具有固定次序的数据元素组成一个序偶
  - 记作 $< x, y >$, 其中的 x 和 y 分别称为第一元素和第二元素
  - 例如:$<$ 中国，亚洲 $>$
  - 三元组可定义为一序偶 $<< x, y >, z >$，其第一元素本身也可是一序偶；可推广到 n 元组的“序偶”
- 序偶关系：任一序偶的集合确定了一个二元关系 R，R 中任一序偶 $< x, y >$ 可记做 xRy
  - $>= \{< x, y > | x, y \in \mathcal{R}, x > y\}$ 表示实数中的“大于”关系
- 其它关系：同属一个集合

数据的逻辑结构



定义与理解

- 从具体问题抽象出来的数据模型；将每个数据元素抽象为一个点，用点间的连边表示数据元素之间的序偶关系；从逻辑关系上描述数据对象。
- 与数据元素本身的形式、内容、相对位置、存储无关。

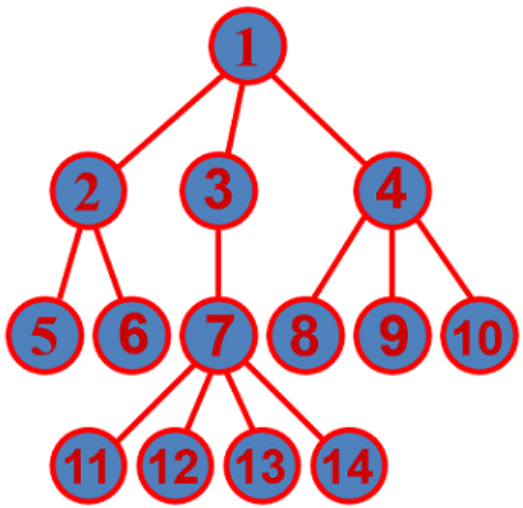
数据的逻辑结构

例子

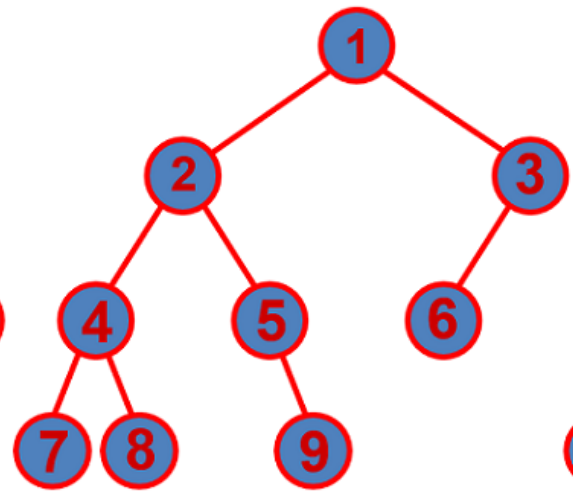
线性结构

树形结构

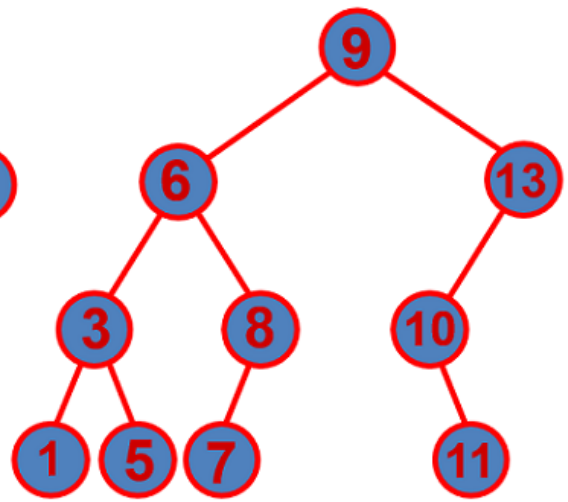
树



二叉树



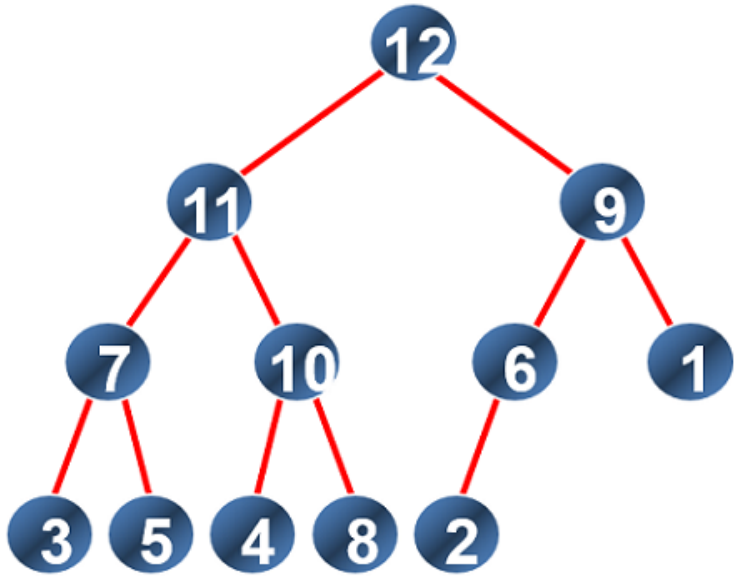
二叉排序树



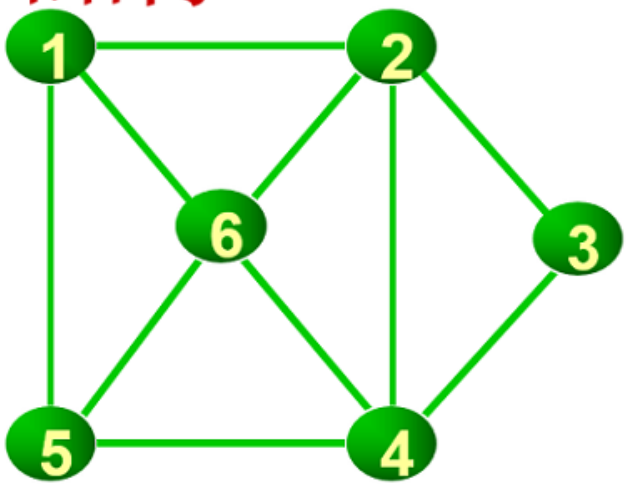
数据的逻辑结构

更多例子

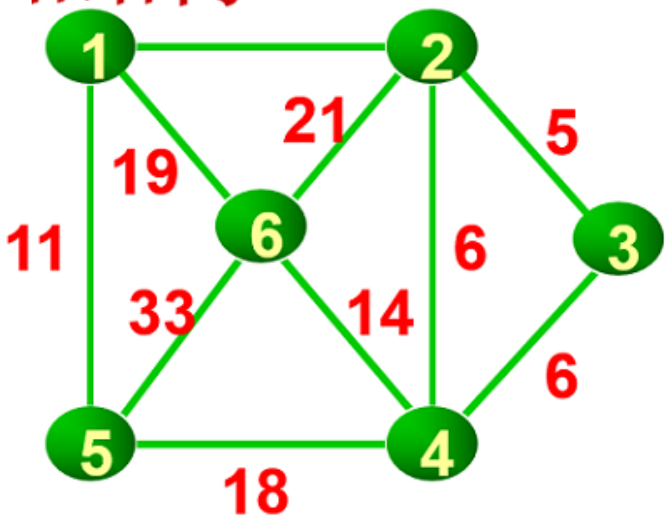
堆结构



图结构



网络结构



数据的存储结构

定义与理解

- 存储结构，即数据结构在计算机中的表示（又称映象），也就是逻辑结构描述的数据如何在线性形式的内存中表示出来
- 包括数据元素的表示和关系的表示
  - 数据元素的表示: 将数据元素转换成 0-1 串，存放在内存中
  - 关系的表示：如何在内存中体现出来?
  - 顺序映象：序偶中两个数据元素相邻体现在内存中的存储位置相邻，信息不丢失
  - 非顺序映象：序偶中两个数据元素相邻，但是在内存中存储位置“可能”不相邻，看上去像是“丢失”了序偶关系

课程学习内容

- 学习数据的逻辑结构，探讨每种逻辑结构如何在计算机内存中的实现（即存储结构）。
- 约束条件或目标：数据占用的内存少，算法处理数据时时空代价小。

从数据类型到抽象数据类型

数据类型

- 一个值的集合和定义在这个值集上的一组操作的总称。
- C 语言中的基本数据类型: int char float double void

抽象数据类型：ADT

- 一个值域以及定义在此值域上的一组操作，记作 ADT
- 例子：矩阵 + (求转置、加、乘、求逆、求特征值) $\Longrightarrow$ 抽象数据类型" 矩阵"
- 在课程与实践中的应用:《数据结构》课程就是学习各种基本的抽象数据类型 ADT 的实现和应用
  - ADT 中的操作，只编写一次，在程序的其他部分或复杂算法，需要时调用操作函数即可;
  - 方便对操作具体功能的修改。

抽象数据类型的描述

ADT 的定义与语法

- 抽象数据类型可用 (D, S, P) 三元组表示, 其中，D 是数据对象，S 是 D 上的关系集，P 是对 D 的基本操作集。
- 定义抽象数据类型的语法：

```
1        ADT 抽象数据类型名{
2          数据对象：  <数据对象的定义>
3          数据关系：  <数据关系的定义>
4          基本操作：  <基本操作的定义>
5        }
```

抽象数据类型的实现

基本步骤

- 选择合适的程序设计语言
- 编程，分别实现数据对象、数据关系和基本操作

例子：ADT 采用 C++ 语言实现的对比

- 抽象数据类型 $\Longrightarrow$ 类 / class
- 数据对象 $\Longrightarrow$ 类中的成员变量
- 基本操作 $\Longrightarrow$ 类中的成员函数/方法

用 C 语言实现 ADT 的注意事项

- 数据元素/项的类型要确定，采用编程语言的基本数据类型来定义，去替换教材上说的“数据元素/项的类型”
- 基本操作仅为示意，不是标准的 C 语言代码，主要差异在参数的表述上。'&' 仅是一个标记，标明其后紧接的变量的值会在基本操作完成后发生改变；不发生改变的变量前不用添加'&'
- 课后阅读 1.3 节 (Page 9-13)，体会伪代码和实际编程实现之间的差异。

抽象数据类型的例子

ADT：三元组

```
1   ADT Triplet {
2   数据对象: D={e1,e2,e3|e1,e2,e3 属于 ElemSet}
        //ElemSet是指定的一个值域，编程时要定义清楚
3   数据关系: R={<e1,e2>,<e2,e3>}
4   基本操作:
5       initTriplet(&T,v1,v2,v3)
6           操作结果: 构造三元组T，元素e1,e2,e3被
7                         分别赋值为v1,v2,v3
8       put(&T,i,v)
9           初始条件: 三元组T已经存在，1<=i<=3
10          操作结果: 改变三元组T的第i个元素的值为v
11      ......                              //更多的基本操作
12  }
```

算法

什么是算法

- 算法的定义：为了解决某类问题而规定的一个有限长的操作序列。

算法的特性

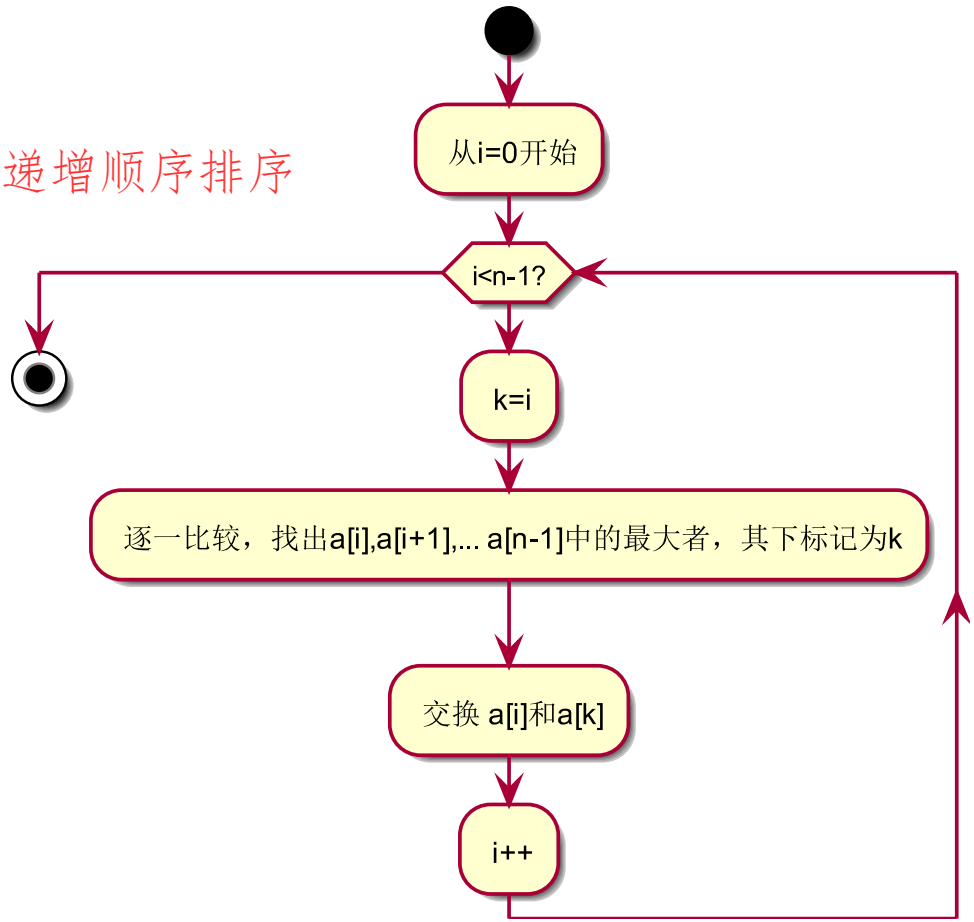
- 有穷性: 算法在执行有穷步后能结束
- 确定性: 每步定义都是确切、无歧义
- 可行性: 每一条运算应足够基本
- 输入: 有 1 个或多个输入
- 输出: 有 1 个或多个输出

算法的例子

选择排序算法的设计过程及结果

- 首先确定解决方案/基本思想：逐个选择最小数据
- 细化核心代码

```
1  void selectSort(int a[],int n){
2  //对n个整数a[0],a[1],…,a[n-1]按递增顺序排序
3    for(int i=0;i<n-1;i++){
4      int k=i;
5      for(int j=i+1;j<n;j++)
6        if (a[j]<a[k])
7          //从a[i]查到a[n-1],
8          //找最小整数,存在a[k]
9          k=j;
10         int temp=a[i];
11         a[i]=a[k];
12         a[k]=temp;
13   }
14 }
```



算法分析

算法分析即评价算法，首先需要评价标准

- 正确性
- 可读性
- 健壮性
- 效率（时间、空间）

评价方法

- 如何分析正确性？机器自动完成正确性验证？
- 如何增加可读性？
- 如何理解健壮性？
- 效率！时间与空间（内存的需求）

时间性能分析

事后统计方法

- 在算法中的某些部位插装时间函数 time ( ), 测定算法完成某一功能所花费时间

存在问题:

- 先运行程序，才能得到结果
- 依赖软硬件环境，比如在一个 cache 大小不一样的机器上运行一个大程序
- 依赖运行时状态，例如内存占用率 100% 时你再去运行一个算法
- 只能在一定程度上度量算法的性能。(OR vs CS)

时间性能分析

事后统计时间的例子（C 语言）

```
1       double start, stop;
2       time (&start);
3       int k = seqsearch (a, n, x);
4       time (&stop);
5       double runTime = stop - start;
6       printf ("%d%d\n",n,runTime );
```

附加说明

- C 语言中关于 time() 的说明：https://baike.baidu.com/item/time.h/4429250

具体实现上述功能的实用工具

- gprof：GNU profiler，记录每个函数的调用次数，每个函数消耗的处理器时间，显示函数的调用关系等
- 工作原理：gcc 编译增加选项：-pg，实现每个函数的入口处插入 _mcount 函数的调用代码，用于统计函数的调用信息：包括调用时间、调用次数以及调用栈信息

时间性能分析

事前估计方法

- 运行时间 = 算法中每条语句执行时间之和
- 每条语句执行时间 = 该语句的执行次数（频度）* 语句执行一次所需时间

存在问题和解决方案

- 语句执行一次所需时间取决于机器的指令性能和速度和编译所产生的代码质量，很难确定
- 设每条语句执行一次所需时间为单位时间，则一个算法的运行时间就是该算法中所有语句的频度之和

时间性能分析

事前估计方法的例子：矩阵相乘

```
1       for ( i =1; i<=n;++i )           // n+1
2         for (j =1;j<=n;++j) {          //n(n+1)
3            c[ i ][ j ] = 0;           //n*n
4            for ( k=1;k<=n;++k)        //n*n*(n+1)
5               c[ i ][ j ] +=a[ i ][ k ]*b[ k ][ j ]; //n*n*n
6         }
```

时间性能分析过程

- 算法执行时间 $T(n)$ 为所有语句的频度之和
  $T(n) = n+1+n(n+1)+n^2+n^2(n+1)+n3 = 2n^3+3n^2+2n+1$
- 运行时间 $\Rightarrow$ 时间复杂度：引入渐进时间复杂度—“O”记号，以体现随问题规模 n 的增长率。
- $T(n) = 2n^3+3n^2+2n+1 \triangleq O(n^3)$
- 算法的时间性能分析是一件非常复杂的、困难的挑战

时间性能分析

事前估计的实际/简化做法

- O 记号的渐进上界 $O(g(n)) : cg(n) \geq f(n)$, 其中 $f(n)$ 是真实的时间复杂度，对应还有渐进下界 Ω、渐进确界 Θ
- 通常我们分析算法的渐进上界就可以评价算法了，为什么？为什么不讨论渐进下界？渐进确界？
- 最坏时间复杂度、最好时间复杂度和平均时间复杂度

不准确的、粗略的估计方法

- 找到嵌套层次最多的循环体，将其最内层中最复杂的语句（函数）挑出来，估计其频度，作为时间复杂度的量级
- 存在问题：
  - 不同的循环体内可能有耗时的、带循环的函数调用，如何确定嵌套最深的循环体？
  - 每个循环的循环次数有差别，最深的循环不一定包括频度最高的语句

时间性能分析

例题: 分析下面算法片段的时间复杂度

```
1         for ( i =1; i<=n ; ++ i)
2               for (j =1; j<=i ; ++j)
3                  for(k =1;k<=j;++k)
4                           x++;
```

分析过程与结果

- 嵌套循环最内层基本语句的执行次数为:

$$\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{j}1=\sum_{i=1}^{n}\sum_{j=1}^{i}j=\sum_{i=1}^{n}\frac{i(i+1)}{2}=\frac{1}{2}\left[\frac{n(n+1)(2n+1)}{6}+\frac{n(n+1)}{2}\right]$$

- 故有 $T(n) = O(n^3)$

时间性能分析

常见时间复杂度

- $O(1), O(n), O(n^2), O(n^b), O(\ln n), O(2^n)$ 分别对应着常数阶、线性阶、平方阶、多项式阶、对数阶、指数阶
- 时间复杂度为 $O(2^n)$ 或 $O(b^n), b > 1$ 时，算法被称为“不可行”的算法，即 n 较大时，算法在可接受的时间内无法运行完成。

练习题

- 两个算法的时间复杂度分别为 $O(\log_{10}(n)), O(\log_2(n))$，那个时间性能更好?
- $2^{100}, (3/2)^n, (2/3)^n, n^n, n^{0.5}, n!, 2^n, \lg n, n \lg n, n^{3/2}, n \lg \lg n, n \lg_2 n, n^2 \lg n, n \lg n^2$ 按增长率排序

时间性能分析

练习题：求斐波那契数

```
1        Long int Fib(int n){
2            if (N<=1)
3                return 1;
4            else return Fib(n-1)+Fib(n-2);
5        } //递归算法
```

完成如下问题：

- 时间复杂度是多少?
- 改进算法，使其成为线性时间复杂度。

时间性能分析

随机置换

- $1 \sim n$ 的自然数保存在数组 $a[0] \sim a[n-1]$ 中，写一个算法，要求生成其随机置换，且每个置换出现的概率相等；
- 给出算法及其时间复杂度分析
- 只考虑时间复杂度最优时，该算法是怎样的？性能如何？
- 参考资料：需要利用 C 语言的库函数 rand():
  “https://baike.baidu.com/item/rand 函数/5916603”

最大子序列和问题

- 给定整数序列 $a_1, a_2, \ldots, a_n$，求 $j > k$ 时，$\sum_{i=k}^{j} a_i$ 的最大值
- 给出算法及其时间复杂度分析，并尝试设计时间复杂度更优的算法
- 如果要求给出 k, j 的值，如何改进算法？

空间性能分析

存储空间的固定部分

- 程序指令代码的空间，常数、简单变量、定长成分 (如数组元素、结构成分、对象的数据成员等) 变量所占空间

可变部分

- 与实例特性有关的成分变量所占空间、引用变量所占空间、递归栈所用空间、通过 new 和 delete 命令动态使用空间

空间复杂度

- 原地工作（额外空间相对输入数据量来说是常数）