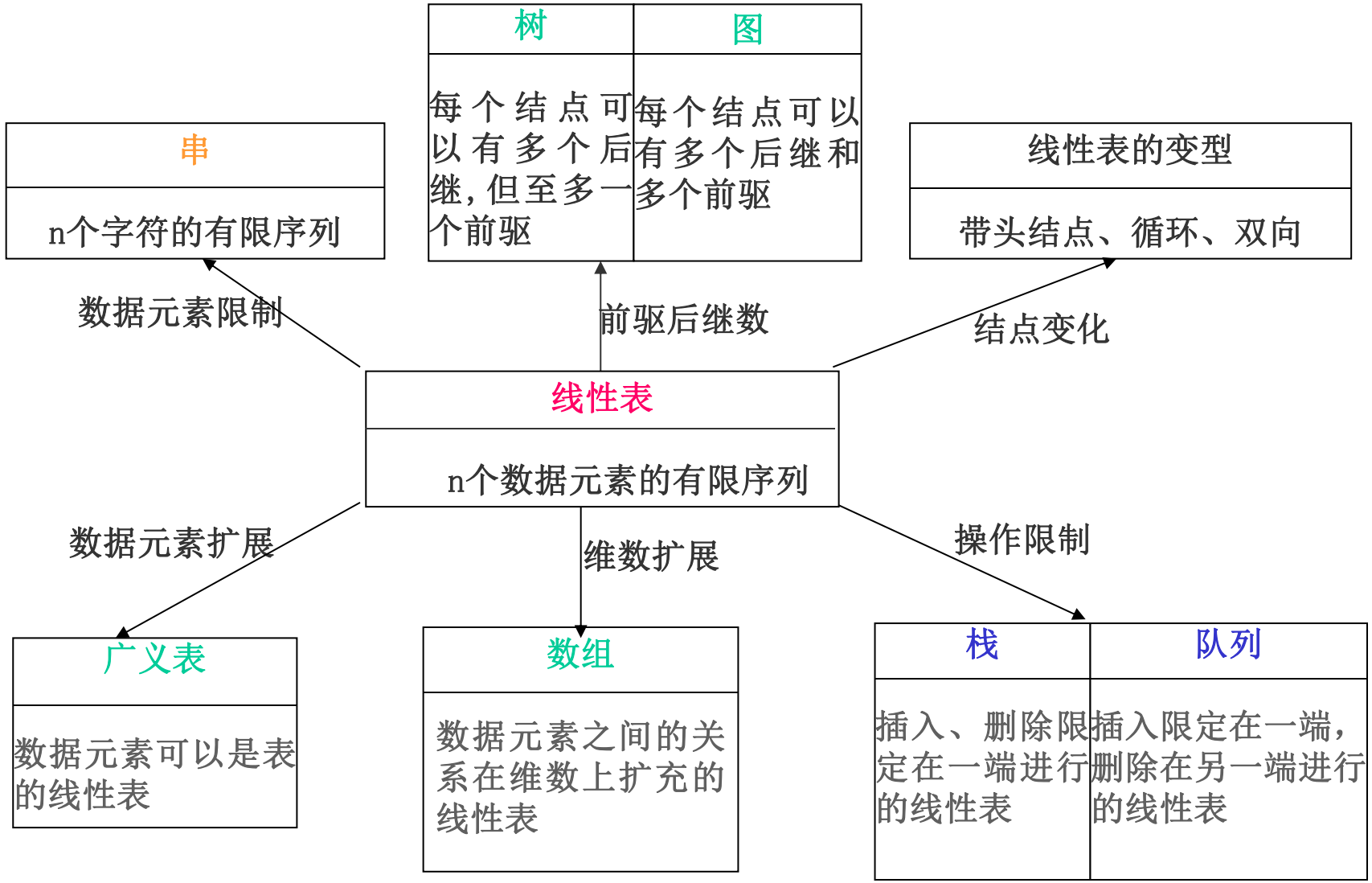
数据结构
Data Structure

2017年秋季学期

刘鹏远


树
图
每个结点可以有多个后继，但至多一个前驱
每个结点可以有多个后继和多个前驱
串
n个字符的有限序列
线性表的变型
带头结点、循环、双向
数据元素限制
前驱后继数
结点变化
线性表
n个数据元素的有限序列
数据元素扩展
维数扩展
操作限制
广义表
数据元素可以是表的线性表
数组
数据元素之间的关系在维数上扩充的线性表
栈
队列
插入、删除限定在一端进行的线性表
插入限定在一端，删除在另一端进行的线性表

算法与算法分析初步

- 算法(Algorithm)：是指令（操作）的有限序列，是对特定问题求解方法(步骤)的一种描述，其中每一条指令表示一个或多个操作。

算法的五大基本特性：

① **有穷性**： 一个算法必须总是在执行有限步之后结束，且每一步都在有限时间内完成。

② **确定性**：算法中每一条指令必须有确切的含义。不存在二义性。

③ **可行性**： 即算法描述的操作都可以通过已经实现的基本运算执行有限次来实现。

④ **输入**： 一个算法有零个或多个输入。

⑤ **输出**： 一个算法有一个或多个输出。

算法vs.程序　设计vs.实现

算法与算法分析初步

查找某元素是否在数组中算法

对数组a从头开始循环：
如 a[i] 等于 x：
　　return 成功
return 失败

```
int  find_x(float a[], float x)
{
    int i;
    for(i=0;i<a.length;i++)
    {
        if(a[i]==x) return 1;
    }
    return 0;
}
```

算法与算法分析初步

- 算法设计 一般是自顶向下，逐步求精

- 对简单选择排序问题select_sort()
  - 明确问题：递增排序
  - 解决方案：逐个选择最小数据

算法与算法分析初步

- 算法框架：

```
for ( int i = 0; i < n-1; i++ )
{
    //从a[i]检查到a[n-1];
    //若最小整数在a[k], 交换a[i]与a[k];
}
```

细化---------->程序

作业：写简单选择排序函数，主函数调用验证

算法与算法分析初步

- 对算法设计的评价角度：

① **正确性**(Correctness)： 应满足具体问题的需求并得到问题的正确答案。（正确性的四个层次）

② **可读性**(Readability)： 应容易供人阅读、交流、理解和修改。

③ **健壮性**(Robustness)： 当输入非法或错误数据时，应能适当地作出反应或进行处理，而不会产生莫名其妙的输出结果。（容错机制）

④ **效率**（Efficiency）： 执行的时间及（存储）空间效率。一般地，这两者与问题的规模有关。

时间短，空间小

算法效率

- 如何度量一个算法的时间效率呢？
  事后统计
  时间：

```
double start, stop;
time (&start);
for(i=0;i<n;i++) select_sort();
time (&stop);
double runTime = stop - start;
cout << runTime/n << endl;
```

上述统计的缺陷

需要运行程序
只得到具体执行时间
效率度量麻烦
影响因素较多
难以比较算法优劣

算法效率

- 考虑影响算法执行时间和空间的因素
  - 机器执行指令的速度（CPU，内存等硬件因素）
  - 书写算法程序的语言
  - 编译程序所产生的机器代码质量
  - 问题的规模
  - 算法

  前三个---外在因素

  问题规模---可用N来表示，规模相同情况下，消除不同算法之间规模因素的差异

  结论：只分析算法自身即可消除其他因素影响

算法效率

- 算法执行时间=算法中所有运算语句的总执行时间
  =算法中每条运算语句执行时间的总和
  =∑**每条运算语句执行的次数*该语句运行时间**

这里的**运算语句---->内置的基本运算语句**

基本运算语句其实执行时间不同

一种简化的方式是假设相同，并**设为单位时间**，则：

算法执行时间=**∑每条运算语句执行的次数**

算法效率

```
int  find_x(float a[], float x)
{
     int i;
     for(i=0;i<n;i++)
     {
          if(a[i]==x)
             return 1;
     }
 }
...
```

每次执行语句次数	执行次数	总次数
0/1	1	0/1
2	n+1	2n+2
1	n	n
1	1	1
总次数		3n+3/3n+4

算法效率

- 算法的执行时间可仅考虑在给定n的情况下，算法执行的基本语句次数，该次数一般来说，是问题规模n的函数。

- 问题：
  - 次数是一个与n相关的函数，则不同函数之间如何比较？

算法效率

次数	算法 C（4n+8）	算法 C'（n）	算法 D（$2n^2+1$）	算法 D'（n^2）
n = 1	12	1	3	1
n = 2	16	2	9	4
n = 3	20	3	19	9
n = 10	48	10	201	100
n = 100	408	100	20 001	10 000
n = 1000	4 008	1 000	2 000 001	1 000 000

算法效率

- 算法的渐近时间复杂度(Asymptotic Time complexity)
  - 简称时间复杂度
  - 大O（big O）
    - T (n) = O(f (n)) 表示存在一个常数C，使得在当n趋于正无穷时总有：

    T (n) ≤ C * f(n)，也就是T(n)/f(n) ≤ 一个常数C

    大O即n趋于无穷时的T(n)变化趋势

    估算大O：

    1、分析算法得到f(n)

    2、忽略系数（因常数C）

    3、忽略低阶项（因该项在n趋于正无穷时与高阶之比为0）

算法效率

- 可知：
  - 算法1：O（n）
  - 算法2：O（n）
  - 算法3：O（n^2）
  - 算法4：O（n^2）

算法效率

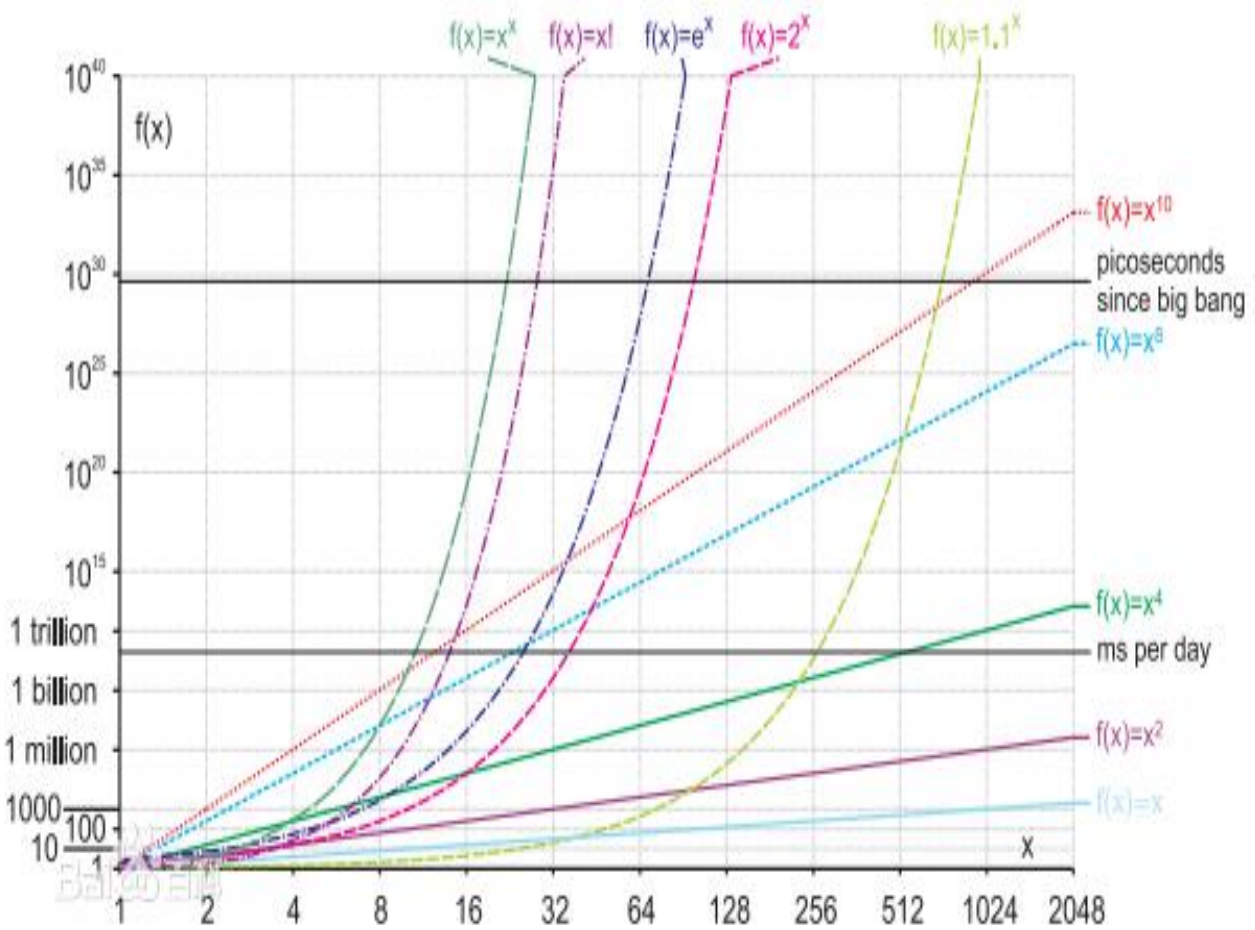
- 常用时间复杂度：

常数阶 对数阶 线性阶 线性对数阶 平方阶 立方阶

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

指数阶

$O(2^n)<O(n!)<O(n^n)$



算法复杂性的不同数量级的变化

n	lon_2n	$nlog_2n$	n^2	n^3	2^n	n!
4	2	8	16	64	16	24
8	3	24	64	512	256	80320
10	3.32	33.2	100	1000	1024	3628800
16	4	64	256	4096	65536	$2.1*10^{13}$
32	5	160	1024	32768	$4.3*10^9$	$2.6*10^{35}$
128	7	896	16384	2097152	$3.4*10^{38}$	∞
1024	10	10240	1048576	$1.07*10^9$	∞	∞
10000	13.29	132877	10^8	10^{12}	∞	∞

算法效率

估算大O，一般仅考虑算法中基本操作的重复的阶

也要掌握一些数列的求和方法

算法分析的加法原理：

针对顺序程序段

T(n, m) = T1 (n) + T2 (m)

由于是有限代码段，因此= O(max（f (n)，g (m))）

算法效率

```
x = 0;  y = 0;
```

T1(n) = O(1)

```
for ( int k = 0; k < n; k ++ )
    x ++;
```

T2(n) = O(n)

```
for ( int i = 0; i < n; i++ )
   for ( int j = 0; j < n; j++ )
      y ++;
```

T3(n) = O(n^2)

T(n) = T1(n)+T2(n)+T3(n) = O(max(1， n， n^2))
= O(n^2)

算法效率

- 算法分析的乘法原理：

针对嵌套程序段

$$T(n, m) = T1(n) * T2(m)$$

$$= O(f(n)*g(m))$$

算法效率

```
void exam ( float x[ ][ ], int m, int n ) {
    float sum [ ];
    for ( int i = 0; i < m; i++ ) {  //x中各行
        sum[i] = 0.0;                    //数据累加
        for ( int j = 0; j < n; j++ )
                sum[i] += x[i][j];
    }
    for ( i = 0; i < m; i++ )  //打印各行数据和
        cout << i << " : " <<sum [i] << endl;
 }
```

渐进时间复杂度为 O((m*n))

算法效率

- 算法分析的分支原理：

针对分支程序段

T(n, m) = P1*T1 (n) + P2*T2 (m)
= O(max（f (n)，g (m))）

算法效率

- 算法的时间复杂度不仅依赖于问题规模n，还与输入实例的初始排列有关。还与多次实例输入排列概率有关。
- 在数组 A[n] 中查找给定值 k 的算法：

```
int i = n-1;
while ( i >= 0 && A[i] != k )
    i--;
return i;
```

算法的语句 i-- 的频度不仅与 n 有关，还与 A[ ] 中各元素的取值，以及 k 的取值有关。算法设计/分析课程中详细分析。

算法效率

- 最好，最坏，平均时间复杂度
- “最好”意义不大
- 一般大O指最坏时间复杂度(或平均时间复杂度)，除非特别说明

算法效率

- 空间复杂度
  - 空间复杂度(Space complexity)：算法编写成程序后在计算机中运行时所需存储空间大小的度量
  - 空间复杂度记为：S(n)=O(f(n))

  - 固定部分
    程序指令代码的空间，常数、变量、输入数据等所占空间
  - 辅助部分
    在算法执行过程中，除固定部分以外，动态使用的空间
  - 一般只考虑算法的辅助部分存储空间。

如何学？老师视角

- 数据结构与算法的形象化，可视化
- 善于用笔，纸思考，善于向两位助教提问
- C语言特别是指针、结构体、函数等要复习好

（上课认真听讲，预习复习写作业这俗套我就不说啦！）

- 时间、时间、时间

如何学？老师视角

- 上机三个层次：

1）照PPT或教材敲入代码，主程序自写，调试成功；

2）看完PPT或教材某个函数，基本自行敲入代码，偶尔翻阅PPT或教材，调试成功；

3）脱稿，自行实现，调试成功。

写在后面

- Donald E. Knuth《计算机程序设计艺术》的问世（1968），将数据结构（编程）科学化系统化
  - 36岁 1974年 图灵奖
  - 媲美牛顿 《自然哲学的数学原理》
  - 使计算机程序设计第一次成为科学

程序=数据结构+算法　Nicklaus Wirth