


课前签到

数据结构与算法2023秋季



 微信扫一扫，使用小程序

1. 微信扫码+**实名**

2. 点击今日签到

签到时间：

9:45~10:15

签到地方：

珠海校区-教学
大楼-C407

人脸识别；智能定位



数据结构与算法 绪论

余建兴

中山大学人工智能学院

提纲

1 课程要求

- 上课纪律
- 考核方法
- 课程书目

2 基本概念

- 数据结构
- 算法
- 选择和衡量

11

课程要求

• 上课纪律



上课期间手机静音：

1. 关闭手机
2. 飞行模式

课程要求

● 考勤

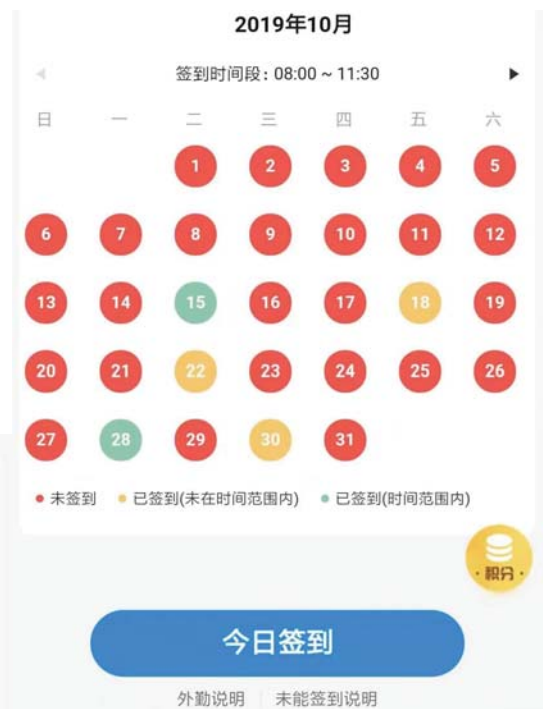
- 不要迟到、早退、旷课
- 有事提前请假
- 课前签到
 - 刷脸+定位
 - 没法代签

签到日期 2022-08-30 至 2023-01-14

签到时间 09:45 至 10:15

签到地址 中大珠海校区教学大楼c503

签到定位 (113.589060,22.348120)中山大学(珠海校区)
教学楼C区



课程要求

● 理论课考核方法

- 10% : 考勤
- 20% : 平时2次作业
- 20% : 期中考试, 第九周的周四随堂考
- 50% : 期末

课程要求

- **实验课考核方法**

- **10%** : 考勤
- **60%** : **3次**课堂限时上级实验
- **30%** : **15次平时**课堂练习

课程要求

- **实验课考核方法**

计分规则:

准确性: 经助教单独检查（比如你不能是改了 autograder.py 的 print 函数）后即可获得 100% 的基础分

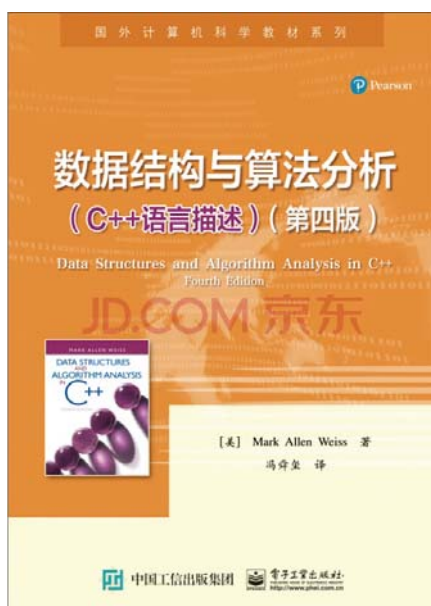
时效性: 在当堂实验课结束前提交的，成绩为：基础分 * 100%；在当堂实验课结束后提交的，成绩为：基础分 * 90%（当天晚上 11 点 59 分 59 秒前），* 80%（第二天晚上 11 点 59 分 59 秒前提交的），* 70%（以此类推）... ...；存在作弊的，成绩为：0

课程要求

- **按时提交作业，严禁抄袭**
 - 所有作业都必须在指定期限内完成并提交
 - 以一个满分为10分的作业为例，计分标准为：
 - 准时提交，满分可达10分(个别加分)
 - 延迟3天内提交，满分可达7分
 - 延迟7天之内提交，满分可达3分
 - 7天之后提交或不交，得分-5分
 - 抄袭得分-20分
 - 提交方式：对分易

课程要求

- **教材**



Mark Allen Weiss(M.A.韦斯)
著，冯舜玺译，《数据结构与算法分析：C++语言描述》第四版，
电子工业出版社，2016年8月

课程要求

● 参考书目

- 许卓群、杨冬青、唐世渭、张铭，《数据结构与算法》，高等教育出版社，2004年7月
- 张铭、赵海燕、王腾蛟，《数据结构与算法--学习指导与习题解析》，高等教育出版社，2005年9月
- 张铭、刘晓丹译，《数据结构与算法分析》(C++第二版、Java版)，电子工业出版社，2002年。(用1998年的第一版也可以，也可以直接用英文原版)
- 许卓群，《数据结构》，高等教育出版社，1988
- 严蔚敏，《数据结构题集》，清华大学出版社

课程要求

● 参考书目

- Donald E.Knuth, The Art of Computer Programming, Addison Wesley. Vol. 1 , Vol. 3. 国防工业出版社影印。(苏运霖译)
- Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest, Clifford Stein, Introduction to Algorithms, MIT Press. 高等教育出版社影印
- William Ford, " Data Structure with C++" , 清华大学出版社

课程要求

- 助教



刘光亚

liugy28@mail2.sysu.edu.cn



江浩维

jianghw9@mail2.sysu.edu.cn

助教和作业沟通QQ群



课程要求

- 实践练习



超算习堂^②
EasyHPC

超算

<https://easyhpc.net/>



矩阵

<https://vmatrix.org.cn/login>



力扣

<https://leetcode-cn.com/>

课程要求

- 课件

数据结构与算法(余建兴)



22级人工智能(AYBSM)

对分易 教学平台

余建兴老师

数据结构与算法

2021年春

[点此编辑»](#) 数据结构与算法

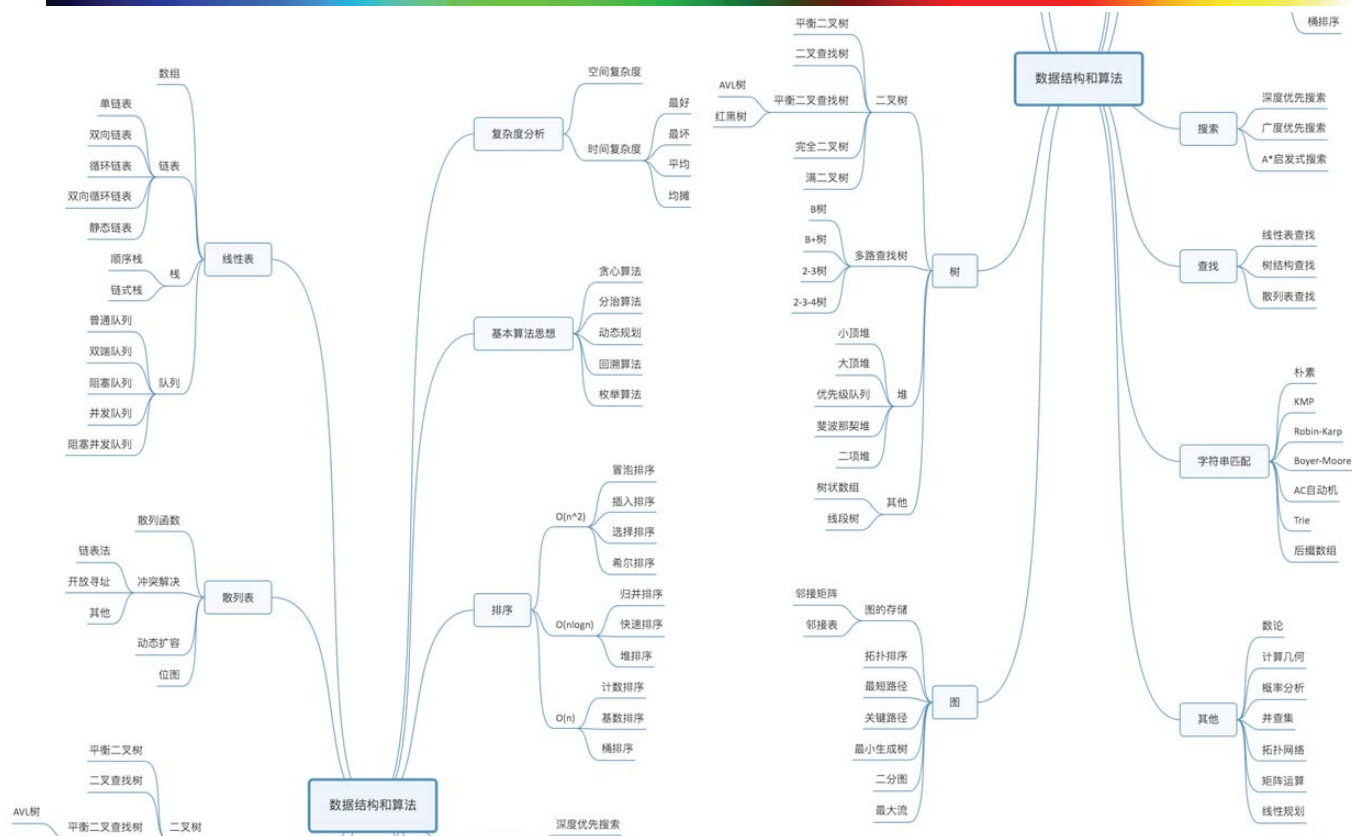
[查看课程简介](#)

<p>班级学生</p>	<p>课程资源</p>	<p>作业</p>	<p>微信消息</p>
<p>考勤</p>	<p>随机分组</p>	<p>手动分组</p>	<p>讨论区</p>
<p>成绩册</p>	<p>在线练习</p>	<p>课堂提问</p>	<p>调查问卷</p>
<p>投票</p>	<p>活动</p>	<p>教学评价</p>	<p>弹幕讨论</p>

课程内容



课程内容



为什么要学这么课呢？

- 它是基础课程
- 考研必考课
- 工作所需的技能

他完成开发并测试通过后，得意地提交了代码。项目经理看完代码后拍着桌子对他说：

“你数据结构是怎么学的？”

“这种实时的排队模块，用什么数据库呀，在内存中完成不就行了吗。赶快改，今天一定要完成，明天一早交给我。”



为什么要学这么课呢？

From another perspective:

- “Computer science is the study of information structures”(Wegner, 1968)



- “Computer science is the study of algorithms” (Knuth, 1968)

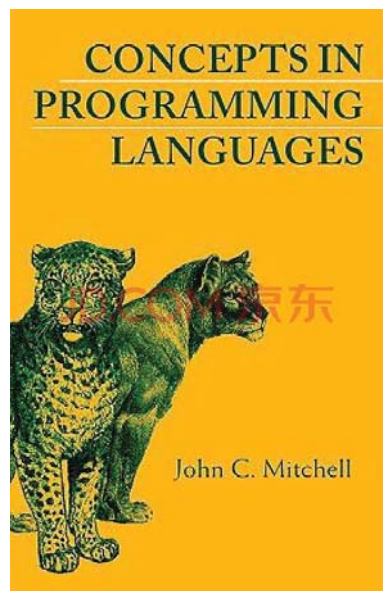
See “Research paradigms in computer science”, Peter Wegner, 1976
数据结构和算法是人工智能领域的核心课题。
两者之间有着本质的联系。

数据结构+算法 = 程序

课程代码

- 使用**伪代码**来展示处理**逻辑**和过程
- 课后实验编程请使用C++代码
- 编程IDE可以使用VStudio，或者Dev-C++

算法思路才是关键，代码只是思路的表达方式



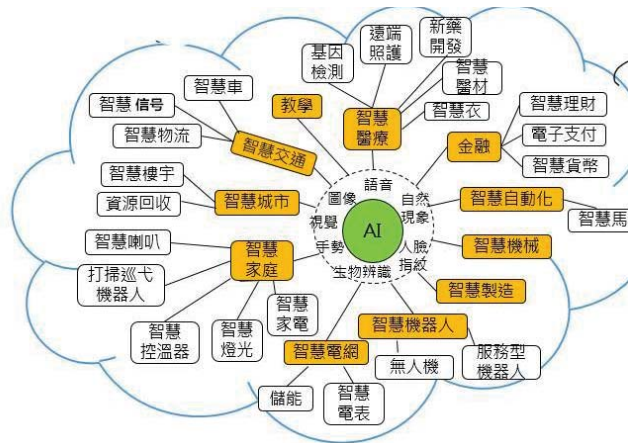
1.1 什么是数据结构

发展历程

- 早期：数值计算



- 现实：业务应用



官方定义

- “数据结构是数据对象，以及存在于该对象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”
 - Sartaj Sahni , 《**数据结构**、**算法**与应用》
- “数据结构是ADT(抽象数据类型，Abstract Data Type)的物理实现”
 - Clifford A.Shaffer , 《**数据结构**与**算法**分析》
- “数据结构(data structure)是计算机中存储、组织数据的方式。通常情况下，精心选择的**数据结构**可以带来最优效率的**算法**”

例1：如何在书架上摆放图书？

如何在书架上摆放图书？

- A. 想好啦
- B. 懒得想，继续吧
- C. 不知道啊
- D. 这个问题不科学

例1：如何在书架上摆放图书？



例1：如何在书架上摆放图书？



例1：如何在书架上摆放图书？

- 图书的摆放要使得2个相关操作方便实现：
 - 操作1：新书怎么插入
 - 操作2：怎么找到某本指定的书

例1：如何在书架上摆放图书？

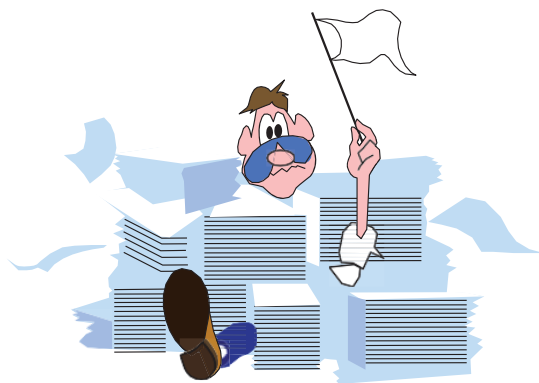
- **方法1：随便放**

- **操作1：新书怎么插入**

- 哪里有空放哪里，一步到位！

- **操作2：怎么找到某本指定的书**

- 一本一本地找.....累死



例1：如何在书架上摆放图书？

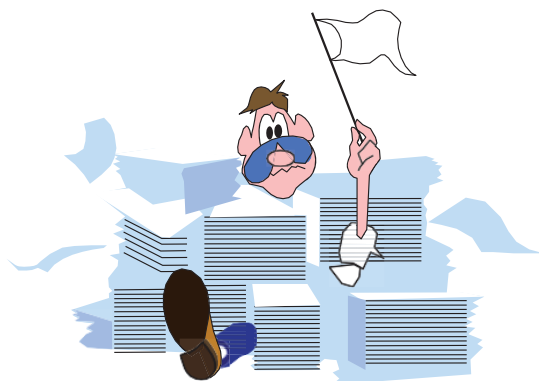
- **方法2：按照书名的拼音字母顺序排放**

- **操作1：新书怎么插入**

- 新进一本《阿Q正传》.....

- **操作2：怎么找到某本指定的书**

- 二分查找！



例1：如何在书架上摆放图书？

- **方法3：把书架划分成几块区域，每块区域指定摆放某种类别的图书；在每种类别内，按照书名的拼音字母顺序排放**
 - **操作1：新书怎么插入**
 - 先定类别，二分查找确定位置，移出空位
 - **操作2：怎么找到某本指定的书**
 - 先定类别，再二分查找
 - **问题：空间如何分配？类别应该分多细？**

**解决问题方法的效率，
跟数据的组织方式有关**

例2：找正整数

- 写程序实现一个函数PrintN，使得传入一个正整数为N的参数后，能顺序打印从1到N的全部正整数

```
void PrintN ( int N )  
{ int i;  
  for ( i=1; i<=N; i++ ){  
    printf("%d\n", i );  
  }  
  return;  
}
```

循环实现

```
void PrintN ( int N )  
{ if ( N ){  
    PrintN( N - 1 );  
    printf("%d\n", N );  
  }  
  return;  
}
```

递归实现

令 N = 100, 1000, 10000, 100000,

例2：找正整数

- 写程序实现一个函数PrintN，使得传入一个正整数为N的参数后，能顺序打印从1到N的全部正整数

```
#include <stdio.h>  
void PrintN ( int N );  
int main ()  
{ int N;  
  scanf("%d", &N);  
  PrintN( N );  
  return 0;  
}
```

解决问题方法的效率， 跟空间的利用效率有关

加和算法对比

关键代码

```
int i, sum = 0, n = 100;
```

```
for (i = 1; i <= n; i++)
```

```
{
```

```
    sum = sum + i;
```

```
}
```

```
printf ("%d", sum) ;
```

```
int sum = 0, n = 100;
```

```
sum = (1 + n) * n / 2;
```

```
printf ("%d", sum) ;
```

数据变化

sum = 1, 3, 6, ...,

sum = 1 + 2 + 3 + ... + 99 + 100

sum = 100 + 99 + 98 + ... + 2 + 1

2×sum = 101 + 101 + 101 + ... + 101 + 101

共 100 个

所以 sum=5050

例3：多项式定点值

- 写程序计算给定多项式在给定点x处的值

$$f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$$

```
double f( int n, double a[], double x )
{ int i;
  double p = a[0];
  for ( i=1; i<=n; i++ )
    p += (a[i] * pow(x, i));
  return p;
}
```

$$f(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_n) \dots))$$

```
double f( int n, double a[], double x )
{ int i;
  double p = a[n];
  for ( i=n; i>0; i-- )
    p = a[i-1] + x*p;
  return p;
}
```

例3：多项式定点值

clock()：捕捉从程序开始运行到clock()被调用时所耗费的时间。这个时间单位是clock tick，即“时钟打点”

常数CLK_TCK(或CLOCKS_PER_SEC)：机器时钟每秒所走的时钟打点数

例3：多项式定点值

- 写程序计算给定多项式 $f(x) = \sum_{i=0}^9 i \cdot x^i$ 在给定点 $x=1.1$ 处的值 $f(1.1)$

```
double f1( int n, double a[], double x )
{ int i;
  double p = a[0];
  for ( i=1; i<=n; i++ )
    p += (a[i] * pow(x, i));
  return p;
}
```

```
double f2( int n, double a[], double x )
{ int i;
  double p = a[n];
  for ( i=n; i>0; i-- )
    p = a[i-1] + x*p;
  return p;
}
```

例3：程序示例

```
#include <stdio.h>
#include <time.h>
#include <math.h>
clock_t start, stop;
double duration;
#define MAXN 10 /* 多项式最大项数, 即多项式阶数+1 */
double f1( int n, double a[], double x );
double f2( int n, double a[], double x );

int main ()
{ int i;
  double a[MAXN]; /* 存储多项式的系数 */
  for ( i=0; i<MAXN; i++ ) a[i] = (double)i;
```

$$f(x) = \sum_{i=0}^9 i \cdot x^i$$

```
}
```

例3：多项式定点值

让被测函数**重复运行**充分多次，使得测出的总的时钟打点间隔充分长，最后计算被测函数**平均每次**运行的时间即可！

**解决问题方法的效率，
跟算法的巧妙程度有关**

归纳

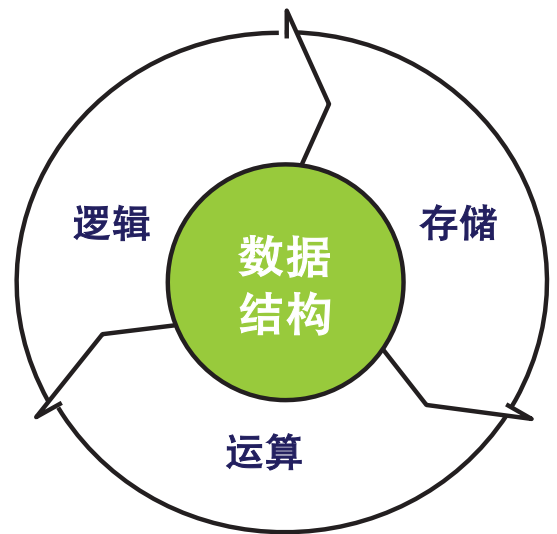
- 即使解决一个非常简单的问题，往往也有**多种方法**，且不同方法之间的**效率可能相差甚远**
- 解决问题方法的**效率**
 - 跟**数据的组织方式**有关(如例1)
 - 跟**空间的利用效率**有关(如例2)
 - 跟**算法的巧妙程度**有关(如例3)

到底什么是数据结构

- **数据对象**在计算机中的组织方式
 - 逻辑结构
 - 物理存储结构
- 数据对象必定与一系列加在其上的**操作**相关联
- 完成这些操作所用的方法就是**算法**

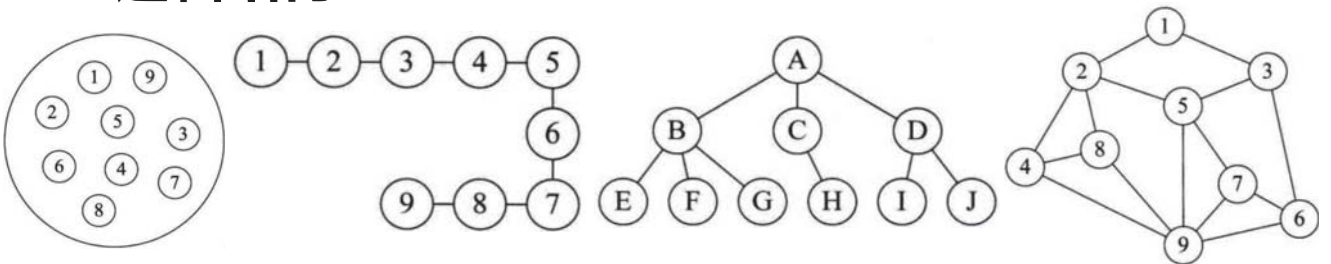
到底什么是数据结构

- **结构:** 实体 + 关系
- **数据结构:**
 - 按照逻辑关系组织起来的一批数据,
 - 按一定的存储方法把它存储在计算机中
 - 在这些数据上定义了一个运算的集合



常用结构

● 逻辑结构



- **线性结构**
 - 线性表 (表, 栈, 队列, 串等)
- **非线性结构**
 - 树 (二叉树, Huffman树, 二叉检索树等)
 - 图 (有向图, 无向图等)
- **图 \supseteq 树 \supseteq 二叉树 \supseteq 线性表**

常用结构

● 物理结构

- 逻辑结构到物理存储空间的**映射**

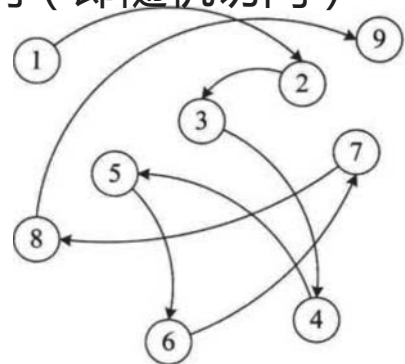
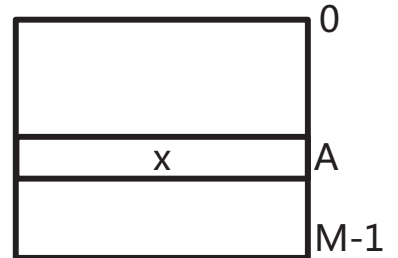
内存

计算机主存储器（内存）

- **非负整数**地址编码，**相邻单元**的集合

- 基本单位是字节

- 访问不同地址所需时间基本相同（即随机访问）



常用结构

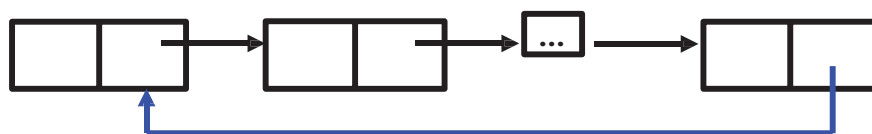
● 物理结构

- 关系元组 $(j_1, j_2) \in r$
(其中 $j_1, j_2 \in K$ 是结点)

- 顺序：存储单元的顺序地址



- 链接：指针的地址指向关系



- 四类：**顺序、链接、索引、散列**

课间习题

- 以下哪种结构是逻辑结构，而与存储和运算无关：
 - A. 数组(array)
 - B. 顺序表(Sequential list)
 - C. 双链表(doubly linked list)
 - D. 队列(queue)

抽象数据类型(Abstract Data Type)

- 数据类型
 - 数据对象集
 - 数据集合相关联的操作集

抽象数据类型示例1

ADT抽象数据类型名称

Data

数据元素之间逻辑关系的定义

Operation

操作1：前进

初始条件

操作结果描述

操作2：后退

....

操作n：跳高

endADT



例子：“矩阵”的抽象数据类型定义

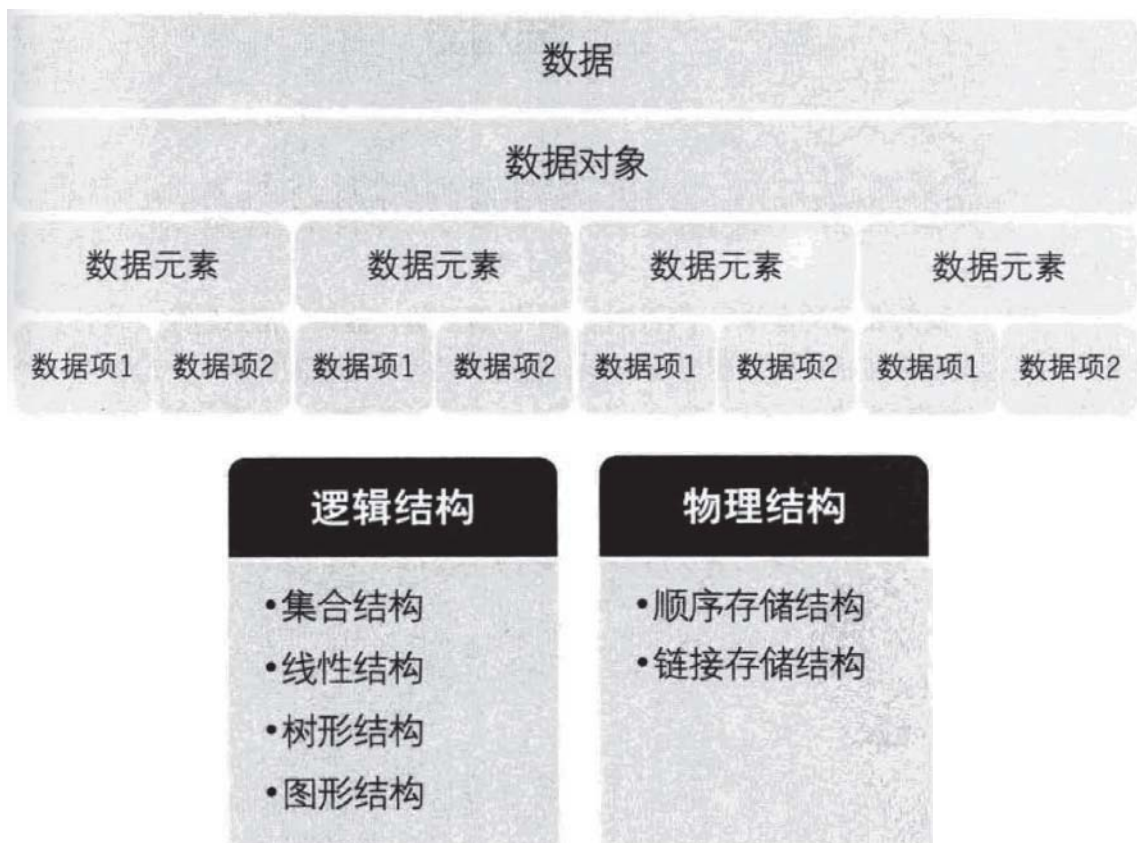
- 类型名称：矩阵(Matrix)

二维数组？一维数组？十字链表？

思考：关于抽象数据类型ADT

- 怎么体现逻辑结构？
- 抽象数据类型等价于类定义？
- 不用模板来定义可以描述 ADT 吗？

小结



习题

- 简述逻辑结构的四种基本关系，并画出它们的关系图

习题

- 简述逻辑结构的四种基本关系，并画出它们的关系图

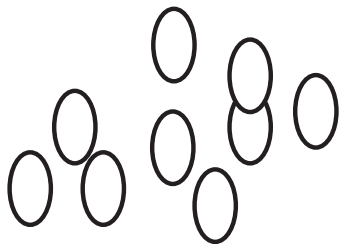
答：

- (1) 集合结构：数据元素之间除了“属于同一个集合”的关系外，别无其他关系。例如，确定一名学生是否为班级成员，只需将班级看作一个集合结构。
- (2) 线性结构：数据元素之间存在一对一的关系。例如，将学生信息数据按照其入学报到的时间先后顺序进行排序，将组成一个线性结构。
- (3) 树结构：数据元素之间存在一对多的关系。例如，在班级的管理体系中，班长管理多个组长，每位组长管理多名组员，从而构成树形结构。
- (4) 图结构或网状结构：数据元素之间存在多对多的关系。例如，多位同学之间的朋友关系，任何两位同学都可以是朋友，从而构成图形结构或网状结构。

其中树结构和图结构都属于非线性结构

习题

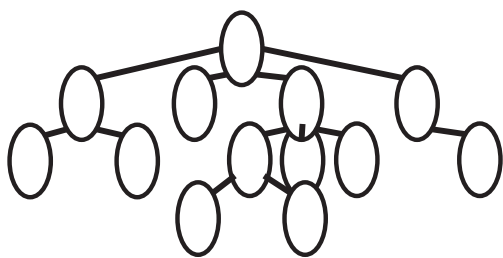
- 简述逻辑结构的四种基本关系，并画出它们的关系图



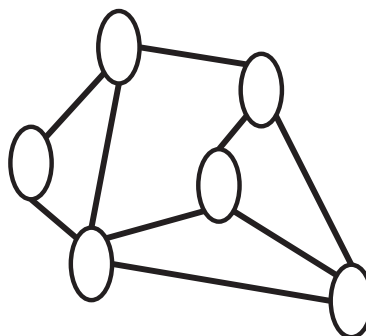
集合结构



线性结构



树结构



图结构

休息时间



第1节休息

1.2 什么是算法

定义

- **算法(Algorithm)**

- 一个有限指令集
- 接受一些输入(有些情况下不需要输入)
- 产生输出
- 一定在有限步骤之后终止
- 每一条指令必须
 - 有充分明确的目标，不可以有歧义
 - 计算机能处理的范围之内
 - 描述应不依赖于任何一种计算机语言以及具体的实现手段

算法有什么用

- 粤省事-健康码：抗疫神器



算法的描述

- 算法的描述可以用下列语言描述

- 框图：便于交流，不依赖于实现；
- 自然语言：易于阅读，但是，不精确，或会有歧义；
- 程序设计语言（程序）：精确，无歧义，但是，不便阅读；
- 介于自然语言和程序设计语言的伪代码；

例1：选择排序算法的伪码描述

```
void SelectionSort ( int List[], int N )  
{ /* 将N个整数List[0]...List[N-1]进行非递减排序 */  
  for ( i = 0; i < N; i ++ ) {  
    从List[i]到List[N-1]中找最小元，并将其位置赋给MinPosition:  
  
    将未排序部分的最小元换到有序部分的最后位置；  
  }  
}
```

课间习题

- 关于算法特性描述正确的有：
 - A. 算法的有穷性指算法必须在有限步骤内结束。
 - B. 组成算法的指令可以有限也可能无限。
 - C. 算法保证计算结果的正确性。
 - D. 算法描述中下一步执行的步骤不确定。

什么是好的算法？

- **正确**：对于合法的输入产生符合要求的输出
- **可读**：易读、便于交流
- **健壮**：考虑适当的错误处理，不会崩溃
- **效率高且内存消耗小**

如何评估效率和复杂度？

- **事后统计**：不同算法的程序运行在同一组输入上，
根据时间和空间的统计来比较优劣。
 - 缺陷：事先编写程序；依赖于程序运行的软硬件环境；
 - 习题：试统计函数`sort()`在不同大小输入上的运行时间
- **事前估计**：一种不考虑算法的程序运行软硬件环境的分析方法。一个算法的运行工作量的大小只依赖于问题的规模

事前估计两类方式

- **空间复杂度 $S(n)$** ——根据算法写成的程序在执行时**占用存储单元的长度**。这个长度往往与输入数据的规模有关。空间复杂度过高的算法可能导致使用的内存超限，造成程序非正常中断。
- **时间复杂度 $T(n)$** ——根据算法写成的程序在执行时**耗费时间的长度**。这个长度往往也与输入数据的规模有关。时间复杂度过高的低效算法可能导致我们在有生之年都等不到运行结果。

例2

```
void PrintN ( int N )
{ if ( N ){
    PrintN( N - 1 );
    printf("%d\n", N );
}
return;
}
```

.....	100000	99999	99998	1
-------	--------	-------	-------	-------------	---	-------

```
PrintN(100000)
PrintN(99999)
PrintN(99998)
PrintN(99997)
.....
PrintN(0)
```

$$S(N) = C \cdot N$$

例3

```
double f( int n, double a[], double x )
{ int i;
  double p = a[0];
  for ( i=1; i<=n; i++ )      (1+2+.....+n)
    p += (a[i] * pow(x, i));  =(n^2+n)/2次乘法
  return p;
}
```

$T(n) = C_1n^2 + C_2n$

```
double f( int n, double a[], double x )
{ int i;
  double p = a[n];
  for ( i=n; i>0; i-- )
    p = a[i-1] + x*p;        n次乘法!
  return p;
}
```

$T(n) = C \cdot n$

什么是好的算法？

- 在分析一般算法的效率时，我们经常关注下面两种复杂度

- 最坏情况复杂度 $T_{worst}(n)$
- 平均复杂度 $T_{avg}(n)$

$$T_{worst}(n) \times T_{avg}(n)$$

复杂度的渐进表示法

示例

- 常数阶

```
int sum = 0, n = 100; /*执行一次*/  
sum = (1+n) * n / 2; /*执行一次*/  
printf (" %d" , sum); /*执行一次*/
```

$O(1)$

- 线性阶

```
int i;  
for(i=0; i<n; i++)  
    {sum = sum + i;}
```

$O(n)$

示例

- 对数阶

```
int count=1, n=100;
while (count < n)
    {count =count*2};
```

$O(\log n)$

- 平方阶

```
int i, j;
for(i=0; i<n; i++)
    { for(j=0; j<n; j++)
        {sum = sum + i;}
    }
```

$O(n^2)$

课间习题

- 已知：平方阶

```
int i, j;
for(i=0; i<n; i++)
    { for(j=0; j<n; j++)
        {sum = sum + i;}
    }
```

$O(n^2)$

```
int i, j;
for(i=0; i<n; i++)
    { for(j=0; j<i; j++)
        {sum = sum + i;}
    }
```

$O(?)$

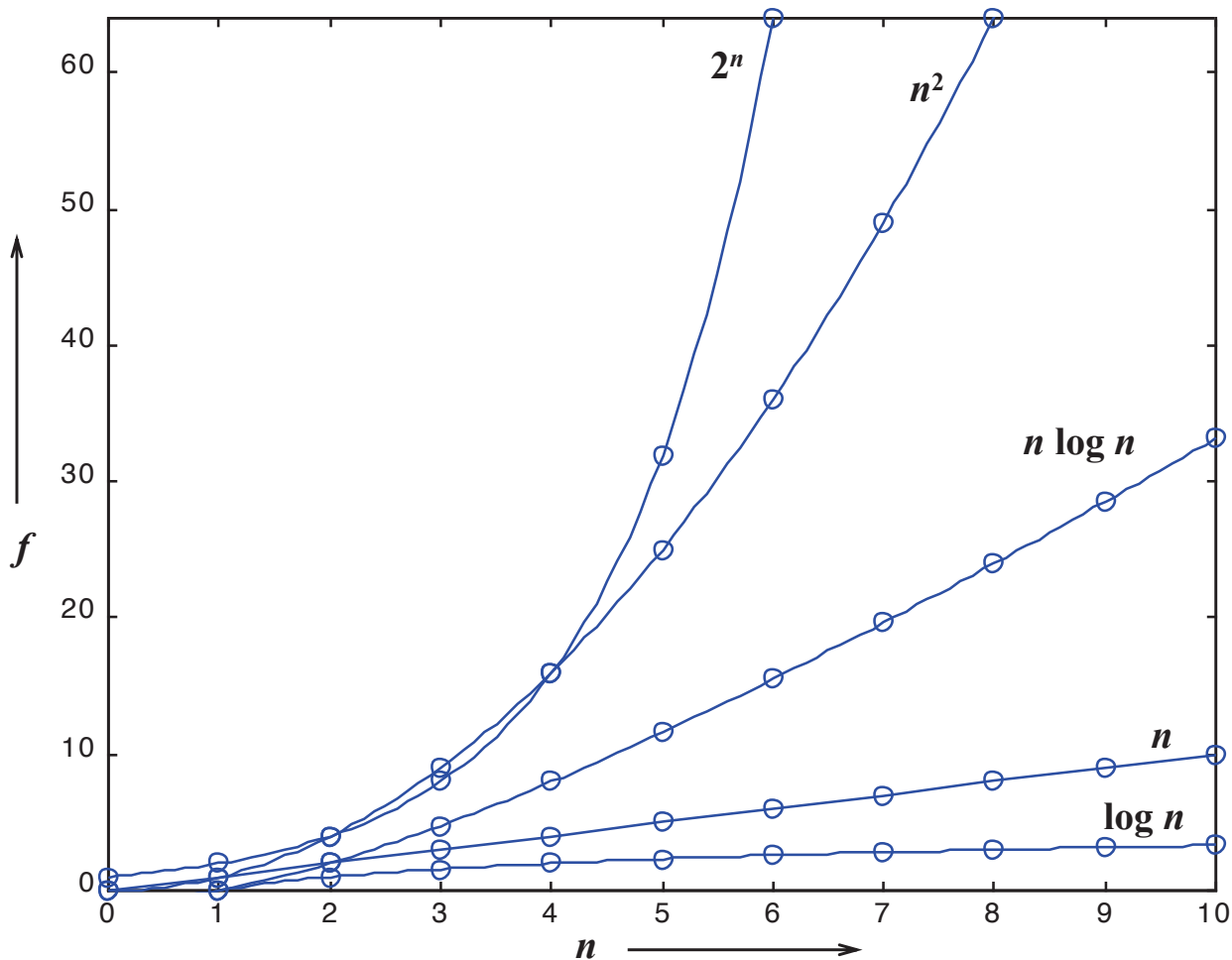
大O表示法

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
$5\log_2 n+20$	$O(\log n)$	对数阶
$2n+3n\log_2 n+19$	$O(n\log n)$	$n\log n$ 阶
$6n^3+2n^2+3n+4$	$O(n^3)$	立方阶
2^n	$O(2^n)$	指数阶

常用函数增长表

函数	输入规模 n					
	1	2	4	8	16	32
1	1	1	1	1	1	1
$\log_2 n$	0	1	2	3	4	5
n	1	2	4	8	16	32
$n \log_2 n$	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32768
2^n	2	4	16	256	65536	4294967296
$n!$	1	2	24	40320	20922789888000	26313×10^{33}

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



每秒10亿指令计算机的运行时间表							
n	$f(n)=n$	$n \log_2 n$	n^2	n^3	n^4	n^{10}	2^n
10	.01 μ s	.03 μ s	.1 μ s	1 μ s	10 μ s	10sec	1 μ s
20	.02 μ s	.09 μ s	.4 μ s	8 μ s	160 μ s	2.84hr	1ms
30	.03 μ s	.15 μ s	.9 μ s	27 μ s	810 μ s	6.83d	1sec
40	.04 μ s	.21 μ s	1.6 μ s	64 μ s	2.56ms	121.36d	18.3min
50	.05 μ s	.28 μ s	2.5 μ s	125 μ s	6.25ms	3.1yr	13d
100	.10 μ s	.66 μ s	10 μ s	1ms	100ms	3171yr	4×10^{13} yr
1,000	1.00 μ s	9.96 μ s	1ms	1sec	16.67min	3.17×10^{13} yr	32×10^{283} yr
10,000	10 μ s	130.03 μ s	100ms	16.67min	115.7d	3.17×10^{23} yr	
100,000	100 μ s	1.66ms	10sec	11.57d	3171yr	3.17×10^{33} yr	
1,000,000	1.0ms	19.92ms	16.67min	31.71yr	3.17×10^7 yr	3.17×10^{43} yr	

μ s = 微秒 = 10^{-6} 秒

ms = 毫秒 = 10^{-3} 秒 sec = 秒

min = 分

hr = 小时

yr = 年

d = 天

课间习题

- 由大到小写出以下时间复杂度的序列： 答案直接写标号，如：(1)(2)(3)(4)(5)

(1) 2^n

(2) $n^{2.5}$

(3) $n(\log_5 n)^4$

(4) $5n^2$

(5) 2^{2^n}

复杂度分析小窍门

- 对给定的算法做渐进分析时，有几个小窍门
 - (1) 若两段算法分别有复杂度 $T_1(n) = O(f_1(n))$ 和 $T_2(n) = O(f_2(n))$
 - 那么两段算法串联在一起的复杂度：
$$T_1(n) + T_2(n) = \max(O(f_1(n)), O(f_2(n)))$$
 - 那么两段算法嵌套在一起的复杂度：
$$T_1(n) \times T_2(n) = O(f_1(n) \times f_2(n))$$
 - (2) 若 $T(n)$ 是关于 n 的 k 阶多项式，那么 $T(n) = \Theta(n^k)$
 - (3) 一个循环的时间复杂度等于循环次数乘以循环体代码的复杂度。例如下面循环的复杂度是 $O(n)$

```
for (i=0; i<N; i++) { x=y*x + z; k++; }
```


复杂度分析小窍门

- 对给定的算法做渐进分析时，有几个小窍门

- (4) 若干层嵌套循环的时间复杂度等于各层循环次数的乘积再乘以循环体代码的复杂度。

例如下列2层嵌套循环的复杂度是 $O(N^2)$ ：

```
for ( i=0; i<N; i++ )  
    for ( j=0; j<N; j++ )  
        { x=y*x + z; k++; }
```

- (5) if-else 结构的复杂度取决于if的条件判断复杂度和两个分支部分的复杂度，总体复杂度取三者中最大。即对结构：

```
if (P1)      /* P1的复杂度为 $O(f_1)$  */  
    P2;      /* P2的复杂度为 $O(f_2)$  */  
else  
    P3;      /* P3的复杂度为 $O(f_3)$  */  
总复杂度为 $\max(O(f_1), O(f_2), O(f_3))$ 
```

课间习题

- 已知一个数组a的长度为n，求问下面这段代码的时间复杂度：

```
for (i=0,length=1;i<n-1;i++){  
    for (j = i+1;j<n && a[j-1]<=a[j];j++)  
        if(length<j-i+1)  
            length=j-i+1;  
}
```

A. $O(n^2)$

B. $O(n)$

C. $\Omega(n)$

D. $\theta(n^2)$

习题

- 分析下面各程序段的时间复杂度

```
i = 1; k = 0;  
while(i < n) { k = k + 10 * i; i++; }
```

```
x = 90; y = 100;  
while(y > 0)  
    if(x > 100)  
        { x = x - 10; y--; }  
    else x++;
```

```
x = n;  
y = 0;  
while(x >= (y + 1) * (y + 1))  
    y++;
```

习题

- 分析下面各程序段的时间复杂度

```
i = 1; k = 0;  
while(i < n) { k = k + 10 * i; i++; }
```

$O(n)$

```
x = 90; y = 100;  
while(y > 0)  
    if(x > 100)  
        { x = x - 10; y--; }  
    else x++;
```

$O(1)$

```
x = n;  
y = 0;  
while(x >= (y + 1) * (y + 1))  
    y++;
```

$O(\sqrt{n})$

1.3 算法分析应用实例

最大子序列和问题

- 给定 N 个整数的序列 $\{A_1, \dots, A_N\}$, 求函数 $f(i, j) = \max(0, \sum_{k=i}^j A_k)$ 的最大值
 - 算法1

```
int MaxSubseqSum1( int A[], int N )
{ int ThisSum, MaxSum = 0;
  int i, j, k;
  for( i = 0; i < N; i++ ) { /* i是子列左端位置 */
    for( j = i; j < N; j++ ) { /* j是子列右端位置 */
      ThisSum = 0; /* ThisSum是从A[i]到A[j]的子列和 */
      for( k = i; k <= j; k++ )
        ThisSum += A[k];
      if( ThisSum > MaxSum ) /* 如果刚得到的这个子列和更大 */
        MaxSum = ThisSum; /* 则更新结果 */
    } /* j循环结束 */
  } /* i循环结束 */
  return MaxSum;
}
```

$$T(N) = O(N^3)$$

最大子序列和问题

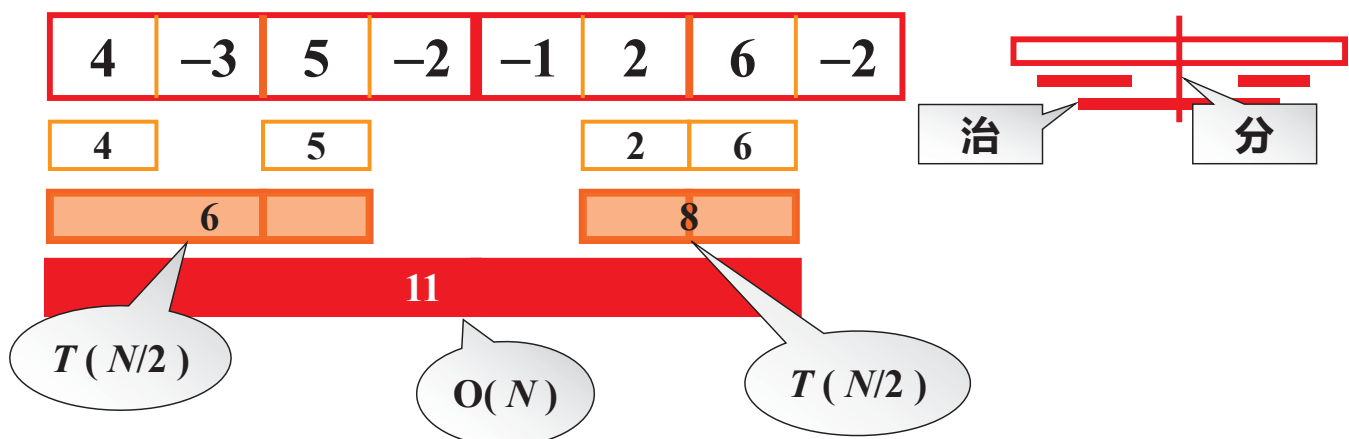
• 算法2

```
int MaxSubseqSum2( int A[], int N )
{ int ThisSum, MaxSum = 0;
  int i, j;
  for( i = 0; i < N; i++ ) { /* i是子列左端位置 */
    ThisSum = 0; /* ThisSum是从A[i]到A[j]的子列和 */
    for( j = i; j < N; j++ ) { /* j是子列右端位置 */
      ThisSum += A[j];
      /*对于相同的i, 不同的j, 只要在j-1次循环的基础上累加1项即可*/
      if( ThisSum > MaxSum ) /* 如果刚得到的这个子列和更大 */
        MaxSum = ThisSum; /* 则更新结果 */
    } /* j循环结束 */
  } /* i循环结束 */
  return MaxSum;
}
```

$T(N) = O(N^2)$

最大子序列和问题

• 算法3：分而治之



$$T(N) = 2 T(N/2) + c N, \quad T(1) = O(1)$$

$$= 2 [2 T(N/2^2) + c N/2] + c N$$

$$= 2^k O(1) + c k N \quad \text{此处 } N/2^k = 1$$

$$= O(N \log N)$$

结论对 $N \neq 2^k$
同样正确

最大子序列和问题

- 算法4：在线处理

```
int MaxSubseqSum4( int A[], int N )
{ int ThisSum, MaxSum;
  int i;
  ThisSum = MaxSum = 0;
  for( i = 0; i < N; i++ ) {
    ThisSum += A[i]; /* 向右累加 */
    if( ThisSum > MaxSum )
      MaxSum = ThisSum; /* 发现更大和则更新当前结果 */
    else if( ThisSum < 0 ) /* 如果当前子列和为负 */
      ThisSum = 0; /* 则不可能使后面的部分和增大, 抛弃之 */
  }
  return MaxSum;
}
```

$T(N) = O(N)$

“在线”的意思是指每输入一个数据就进行即时处理，在任何一个地方中止输入，算法都能正确给出当前的解

序列A[] 仅需扫描一遍！

最大子序列和问题

- 上述4种算法用于求最大子列和所需的运行时间的比较

单位：秒

算法		1	2	3	4
时间复杂性		$O(N^3)$	$O(N^2)$	$O(N \log N)$	$O(N)$
子列 大小	$N=10$	0.00103	0.00045	0.00066	0.00034
	$N=100$	0.47015	0.01112	0.00486	0.00063
	$N=1,000$	448.77	1.1233	0.05843	0.00333
	$N=10,000$	NA	111.13	0.68631	0.03042
	$N=100,000$	NA	NA	8.0113	0.29832

注：不包括输入子列的时间。

NA – Not Acceptable, 不可接受的时间

如何选择和评价算法

- 仔细分析所要解决的问题，特别是求解问题所涉及的数据类型和数据间逻辑关系
- 数据结构的初步设计往往在算法设计之先
- 注意数据结构的可扩展性。包括考虑当输入数据的规模发生改变时，数据结构是否能够适应。同时，数据结构应该适应求解问题的演变和扩展
- 数据结构的设计和选择也要比较算法的时空开销的优劣

算法的度量

- 数据结构
 - 一定的空间来存储它的每一个数据项
 - 一定的时间来执行单个基本操作
- 代价和效益
 - 空间和时间的限制
 - 程序设计工作量

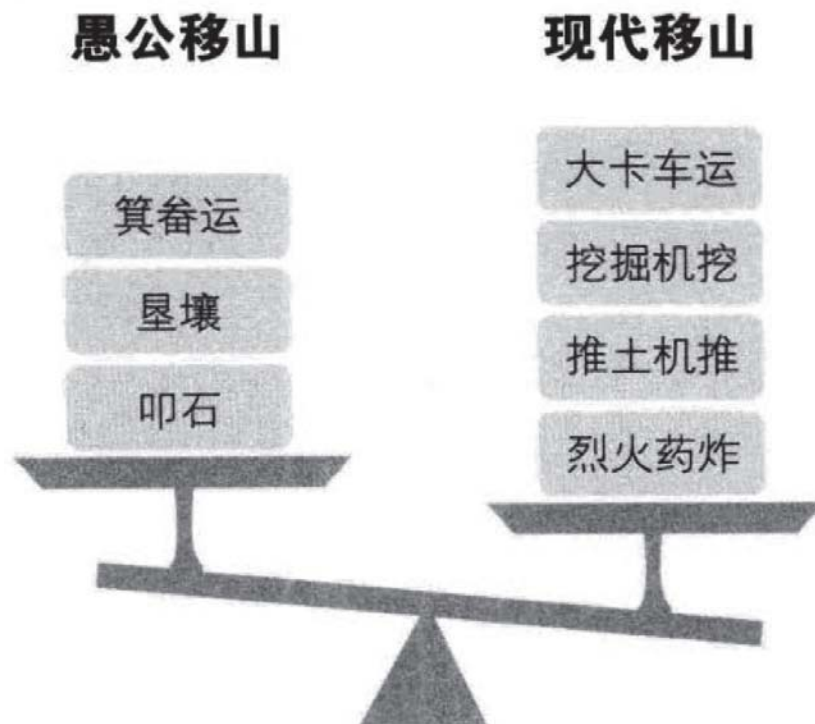
时空资源的折中原理

- 对于同一个问题求解，一般会存在多种算法。而这些算法在时空开销上的优劣往往表现出“**时空折中**”的性质
- “时空折中”，是指为了改善一个算法的时间开销，往往可以通过增大空间开销为代价，而设计出一个新算法来
- 有时也可以为了缩小算法的空间开销，而牺牲计算机的运行时间，通过增大时间开销来换取存储空间的节省

小结

- 数据结构的地位与重要意义
- 数据结构的主要内容
- 抽象数据类型的概念
- 算法及其特点
- 算法的有效性度量
- 数据结构的选择

小结



算法是一门艺术

- 一流程程序员靠**数学**
- 二流靠**算法**
- 三流靠**逻辑**
- 四流靠**SDK**
- 五流靠**Google和StackOverFlow**
- 六流靠**百度和CSDN**

低端的看高端的就是黑魔法！

休息时间



第2节休息

谢谢！