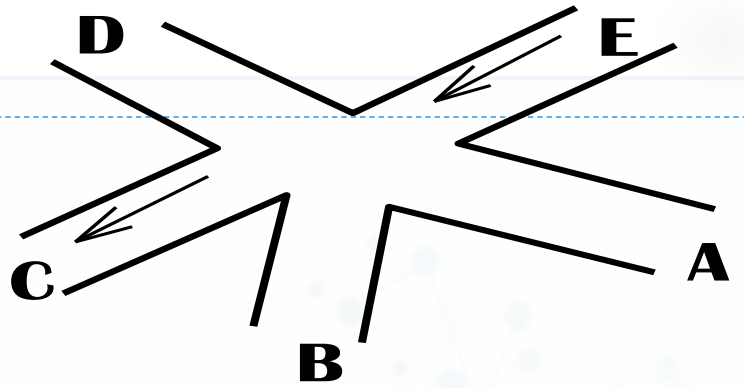
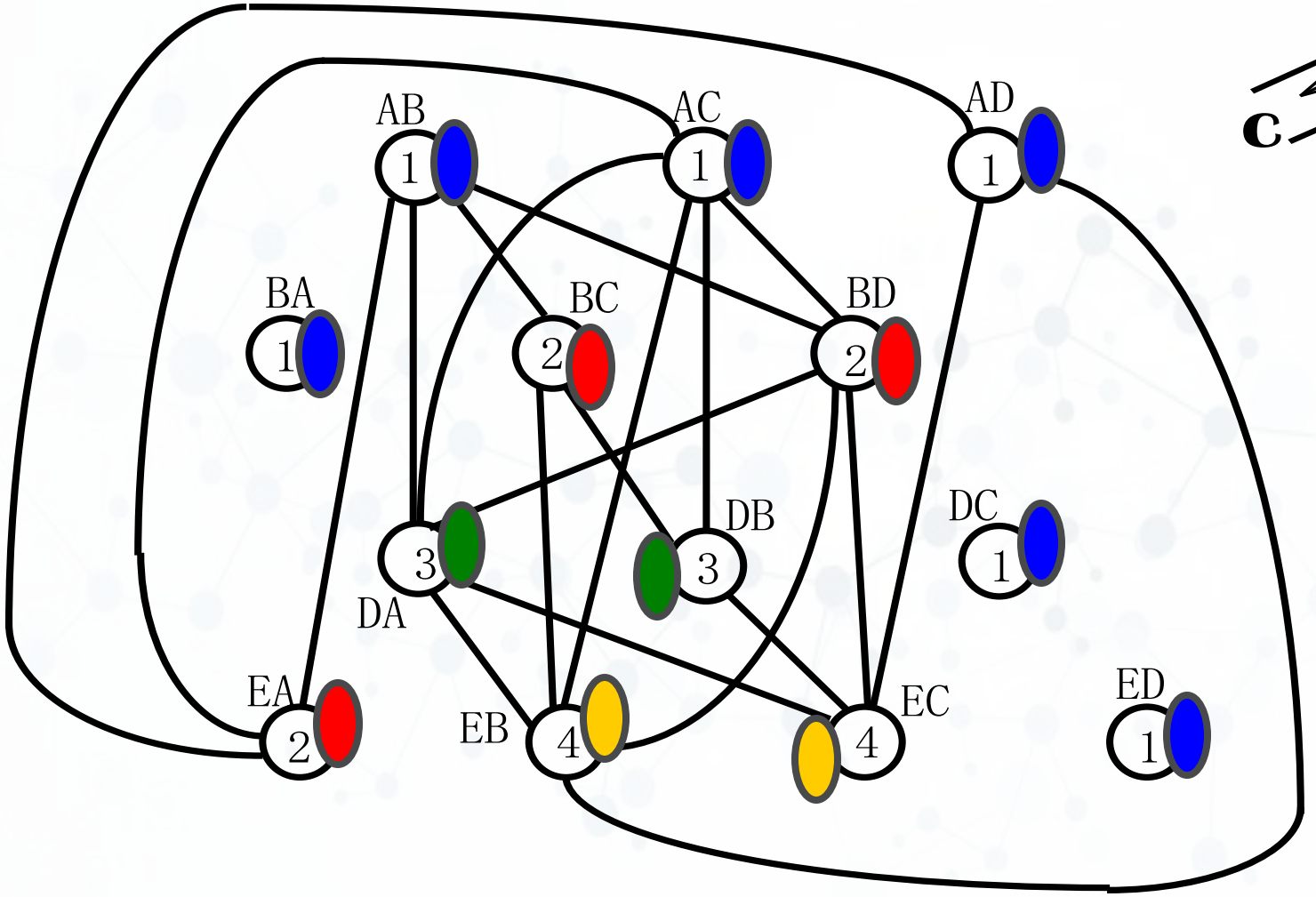
数据结构与算法（A）-W01/概述

北京大学 陈斌斌

2024.09.11

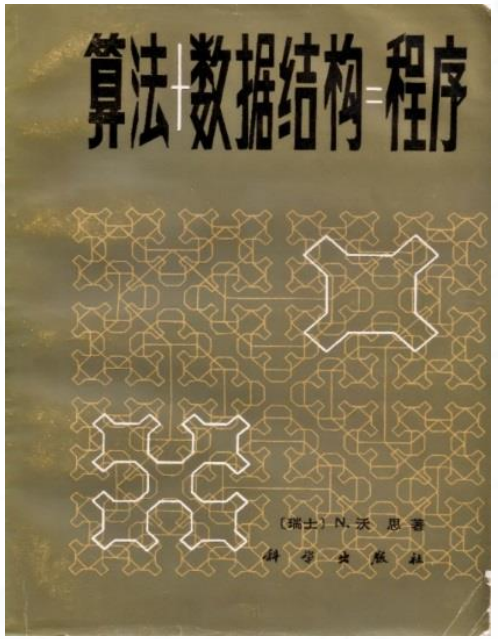
引 子





算法 + 数据结构 = 程序

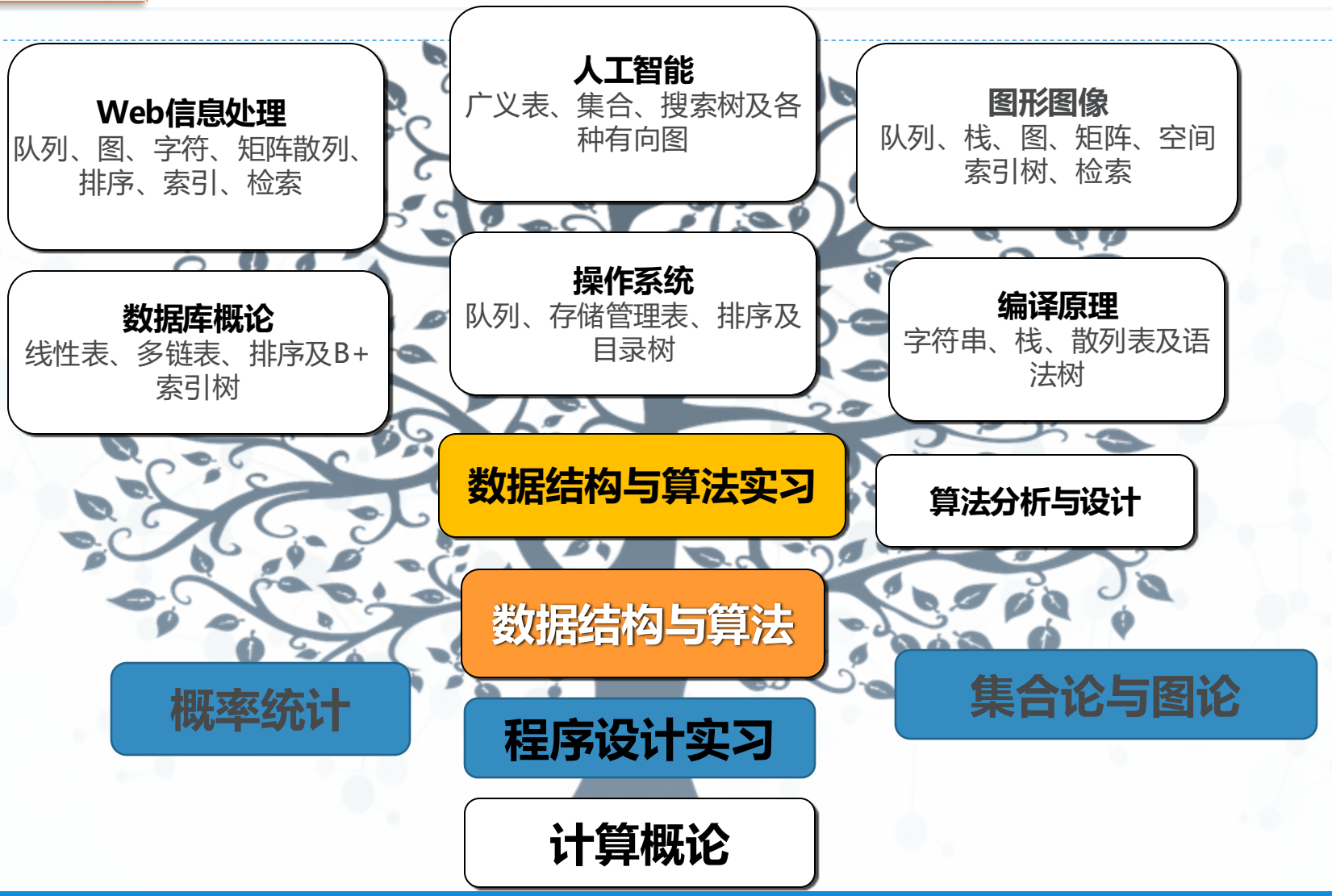
- **问题（problem）**：从输入到输出的一种映射函数
- **数据结构（Data Structure）**：逻辑数据结构在计算机中的存储表达，支持相应的操作
- **算法（algorithm）**：对特定问题求解过程的描述方法
- **程序（program）**：算法在计算机程序设计语言中的实现



教学目标

- **“数据结构 + 算法 = 程序”**
  - 基本的抽象数据类型（ADT） 及其应用
  - 合理组织数据，有效表示数据，有效处理数据
  - 算法的设计分析技术
- **抽象能力**
  - **问题 —— 数据 —— 算法**
- 提高**程序设计的质量**

地　位


Web信息处理
队列、图、字符、矩阵散列、排序、索引、检索
人工智能
广义表、集合、搜索树及各种有向图
图形图像
队列、栈、图、矩阵、空间索引树、检索
数据库概论
线性表、多链表、排序及B+索引树
操作系统
队列、存储管理表、排序及目录树
编译原理
字符串、栈、散列表及语法树
数据结构与算法实习
算法分析与设计
数据结构与算法
概率统计
集合论与图论
程序设计实习
计算概论

数据结构与算法

—— PKU MOOC课程

- 选课对象：大二 **理工科** 学生
- 主讲人**张铭:** 北京大学信息科学技术学院教授，博士生导师，ACM Education Council中国委员兼ACM中国教育专委会主席
- 教材：**《数据结构与算法》**。张铭，王腾蛟，赵海燕 编，高等教育出版社，2008年6月。——普通高等教育“十一五”国家级规划教材（入选“十二五”）
- 国家精品课网站

http://jpk.pku.edu.cn/course/sjjg/

课程结构

第1章 概论

（概念、逻辑结构、存储结构，ADT，算法特征，**算法量度**）

第2章 线性表

（线性表ADT和存储结构）

第3章 栈和队列

（栈的存储和应用，栈和表达式，**栈和递归，**队列的概念、循环队列）

第4章 字符串

（字符串概念、ADT、**模式匹配**、KMP算法）

第5章 二叉树

（二叉树的概念和 ADT，二叉树的深度和宽度遍历，**二叉树的非递归后序周游**，二叉树实现，二叉搜索树、堆，Huffman 树）

第6章 树

（树的基本概念，树的深度和宽度遍历，**树的顺序存储**）

第7章 图

（图的概念，图的遍历和存储，图的拓扑排序，**图的单源最短路径 Dijkstra 算法**，图的 **Floyd 算法**，**最小支撑树**的 Prim 算法和 Kruskal 算法）

第8章 内排序

（内排序基本概念，插入排序、冒泡排序和选择排序等简单排序，shell排序，**快速排序**，归并排序，**堆排序**、**桶式排序**，**基数排序**、**地址排序**）

第9章 文件管理和外排序

（**文件基本概念、置换选择排序、选择树、败方树，多路归并**）

第10章 检索

（检索的基本概念，顺序检索，**集合检索**，**散列函数**，开散列法，**闭散列，探测算法**）

第11章 索引

（索引基本概念，线性索引，**倒排索引**，**B/B+树，红黑树**）

第12章 高级数据结构（**了解**，不要求掌握）

（多维数组，矩阵，广义表，**Trie树**，最佳二叉搜索树，**AVL树，自组织伸展树**）

前沿应用：搜索引擎、社会网络与数据挖掘……

基础：

抽象数据类型ADT

算法分析
时空折衷

理论 抽象 设计

逻辑	运算	存储
线性（表、栈、队列、串）	排序：插入、分治、快速、堆、基数	顺序、链接、散列、索引
树(二叉树、森林)	检索：二分、散列	内存、外存
图（有向、无向、DAG）	索引： BST、B+	外排序 B+树，倒排

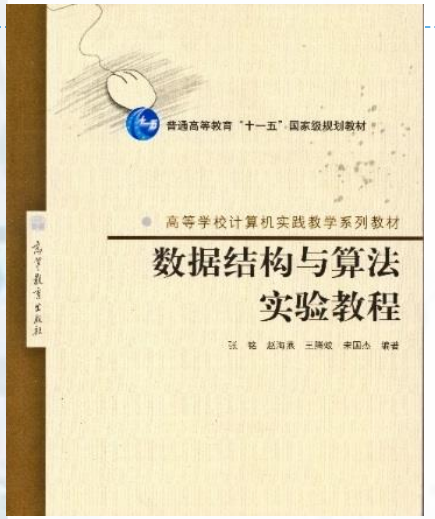
扩展研究：广义表，稀疏矩阵，字符树，Patricia树，AVL， 红黑树，伸展树 ……

课程评分标准

- 平时 40 %
  - MOOC 15
  - POJ 10
  - 书面作业 15
- 考试 60 %
  - 期中考试 20
  - POJ上机考试 15
  - 期末考试 25

慕课链接
https://www.icourse163.org/course/PKU-1002534001

配套教材



- 张铭，赵海燕，王腾蛟，**《数据结构与算法实验教程》**，高等教育出版社，2011年 1月。普通高等教育“十一五”国家级规划教材

- 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 10月。——“十五”国家级规划教材配套参考书

数据结构与算法参考网站

- 1. 北大课程网 http://course.pku.edu.cn
  - 高级数据结构书面作业（可以交作业本，或者电子版）
- 2. 国家精品课程
  http://jpk.pku.edu.cn/course/sjjg/
- 3. Berkeley《数据结构》
  http://www.cs.berkeley.edu/~jrs/61b/
- 4. MIT 的《算法导论》（有OCW链接）
  http://stellar.mit.edu/S/course/6/sp13/6.006/
- 5. Stanford 算法 https://www.coursera.org/course/algo
- 6. Princeton 算法课
  https://www.coursera.org/course/algs4partI
- 7. Web 上的术语资源 http://www.nist.gov/dads/

高级数据结构参考网站

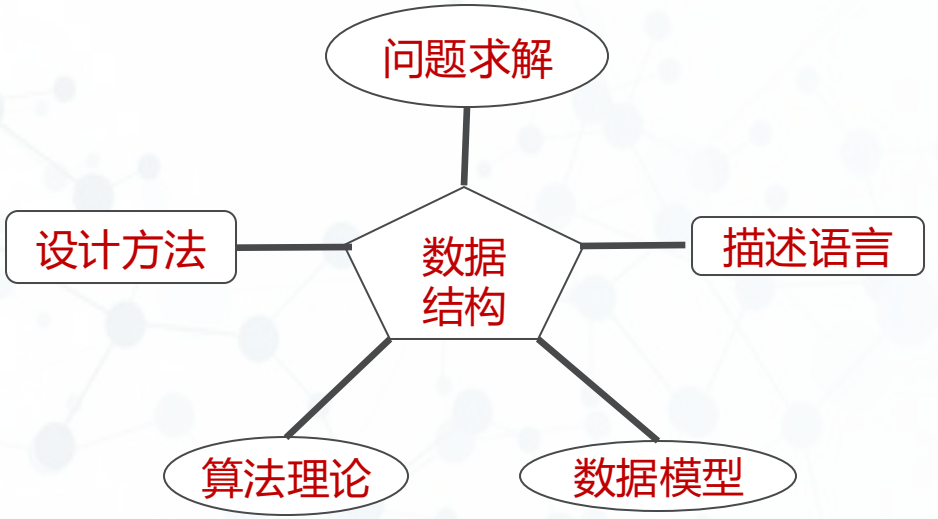
- Algorithms in the Real World（后缀树， 加密算法，整数规划，生物信息学等）
  - http://www.cs.cmu.edu/~guyb/realworld.html
- Advanced Data Structures and Algorithms（自适应数据结构、树堆、随机算法、流数据处理、关联规则）
  - http://theory.stanford.edu/~rajeev/cs361.html
- Advanced Data Structures（后缀树/后缀数组，自适应数据结构，高级Trie等）
  - http://www.cs.biu.ac.il/~moshe/ds1.html

第1章　概论

- 问题求解
- 数据结构及抽象数据类型
- 算法的特性及分类
- 算法的效率度量

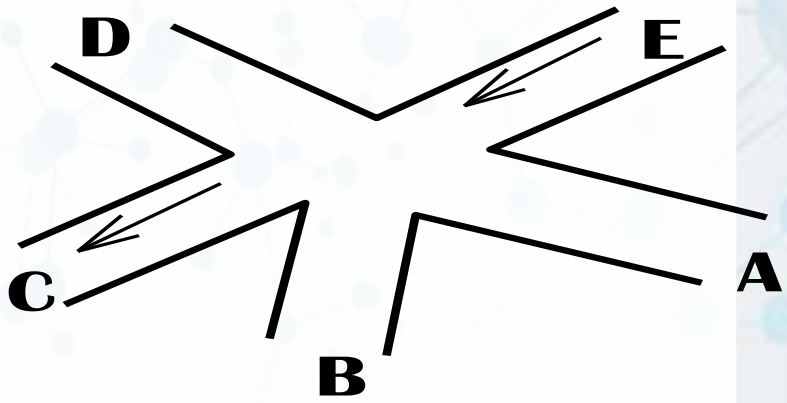
1.1 问题求解

- 编写计算机程序的目的?
  - 解决实际的应用问题
- 问题抽象
  - 分析和抽象任务需求，建立问题模型
- 数据抽象
  - 确定恰当的数据结构表示数学模型
- 算法抽象
  - 在数据模型的基础上设计合适的算法
- 数据结构 + 算法，进行程序设计
  - 模拟和解决实际问题



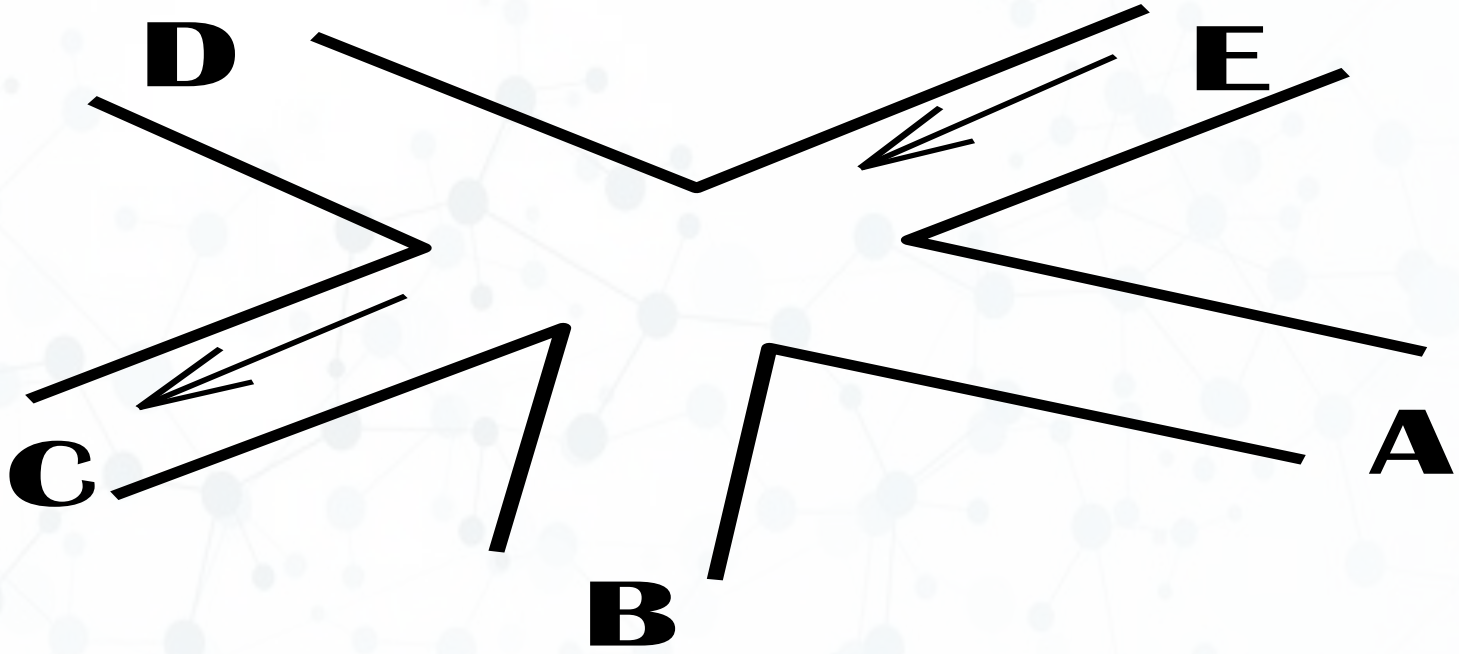
多叉路口交通灯管理问题

- 五叉路口
  - 右行规则
  - 道路C、E是箭头所示的单行道
- 把可以同时行驶而不发生碰撞的路线用一种颜色的交通灯指示
- 用多少种颜色的交通灯，怎样分配给这些行驶路线？
  - 颜色越少则管理效率越高
  - 不考虑过渡灯（例如黄灯）



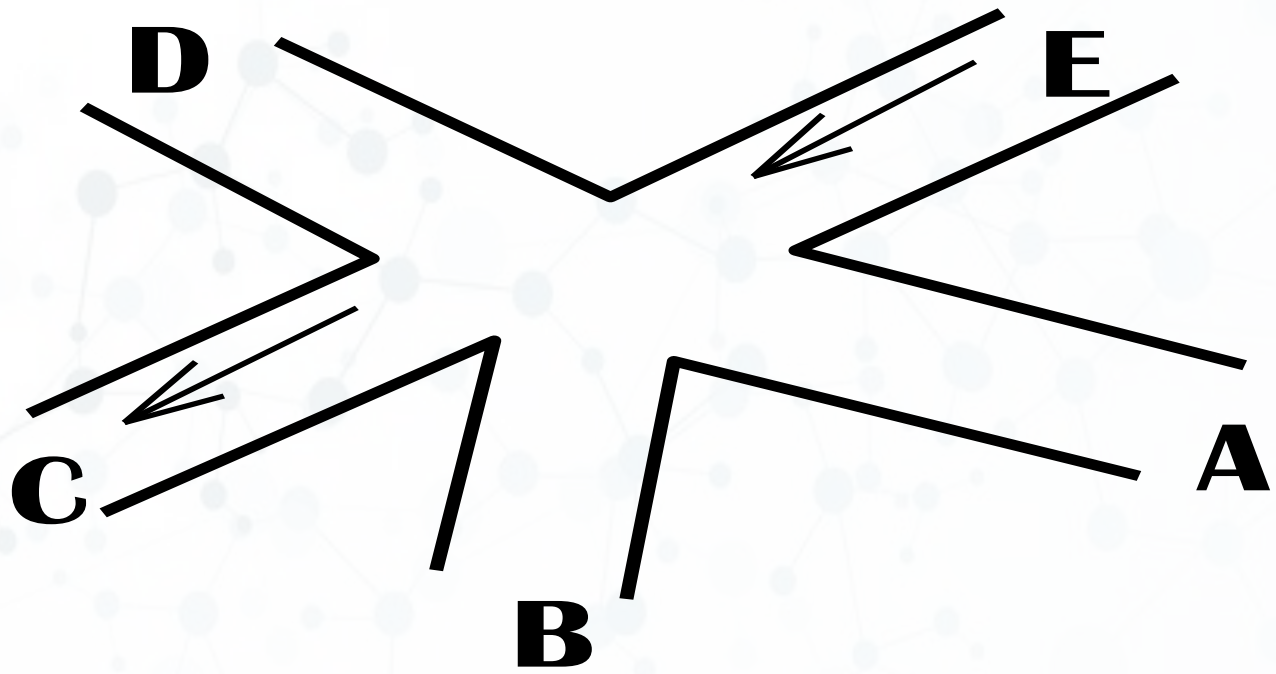
13种行驶路线

- AB,AC,AD
- BA,BC,BD
- DA,DB,DC
- EA,EB,EC
- ED
- **不能同时**
  - 如AB、BC；EB、AD
- **可以同时**
  - 如AB、EC

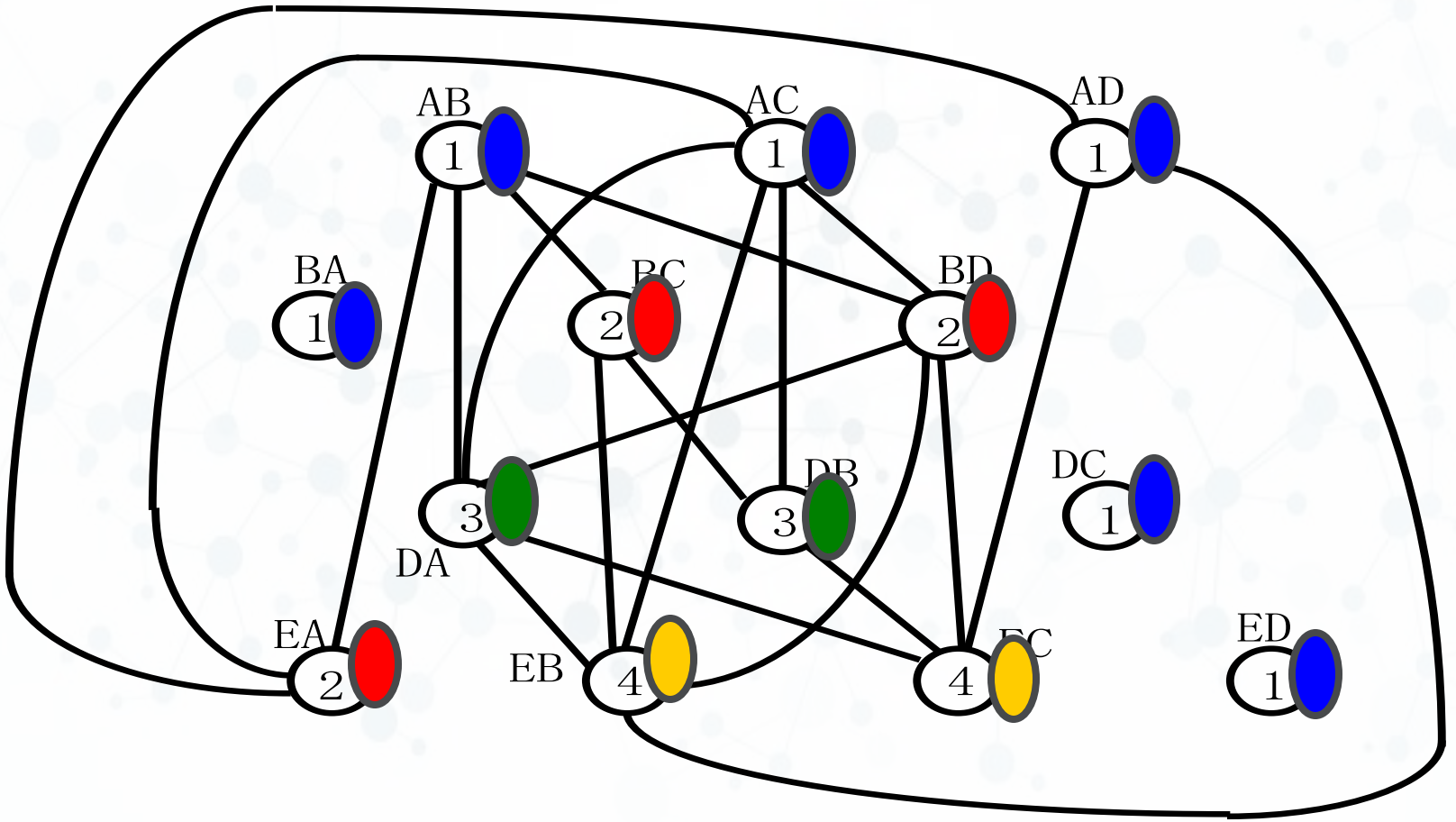


不能同时走的路线20对

- (AB BC) (AB BD) (AB DA) (AB EA)
- (AC DA) (AC BD) (AC DB) (AC EA) (AC EB)
- (AD EA) (AD EB) (AD EC)
- (BC EB) (BC DB)
- (BD DA) (BD EB) (BD EC)
- (DA EB) (DA EC)
- (DB EC)



有连线则不能同时通行



北京大学 陈斌 gischen@pku.edu.cn 2024

平面图着色（例为美国地图）

- 对国家区域平面图进行着色
  — 要求相邻的区域用不同的颜色

平面图着色问题

- 最少着色分组的最优解问题是NP复杂性问题
  - 穷举法或回溯法来解决地图着色问题。对于小型地图可以使用
  - 对于大型模式，由于时间的指数上升，不可接受

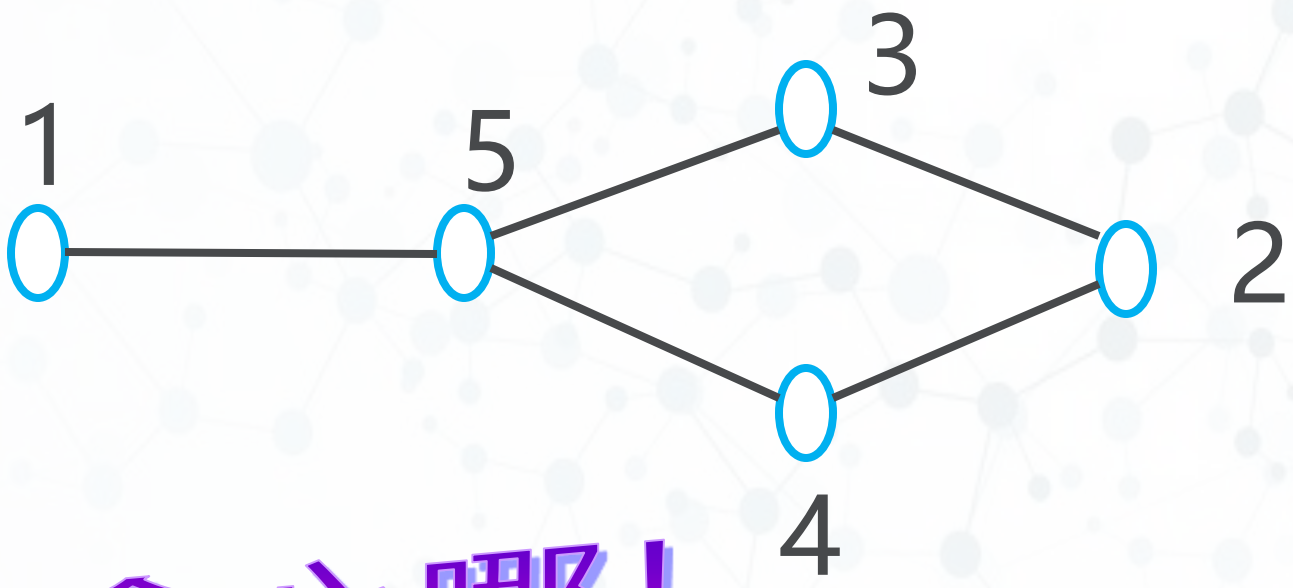
- 指数级或NP的难解问题往往通过逼近方法来求近似最优解

平面图着色：贪心法

- 1. 用一种新颜色给尽可能多的顶点着色
  - (1) 选择一个未着色的顶点并着该颜色
  - (2) 扫描其他未着色的点，考察其邻接点是否着了该颜色，若没有就着该颜色
- 2. 换一种颜色重复1，
  直到所有顶点全部着色为止

贪心法近似解

- 按1，2，3，4，5顺序着色



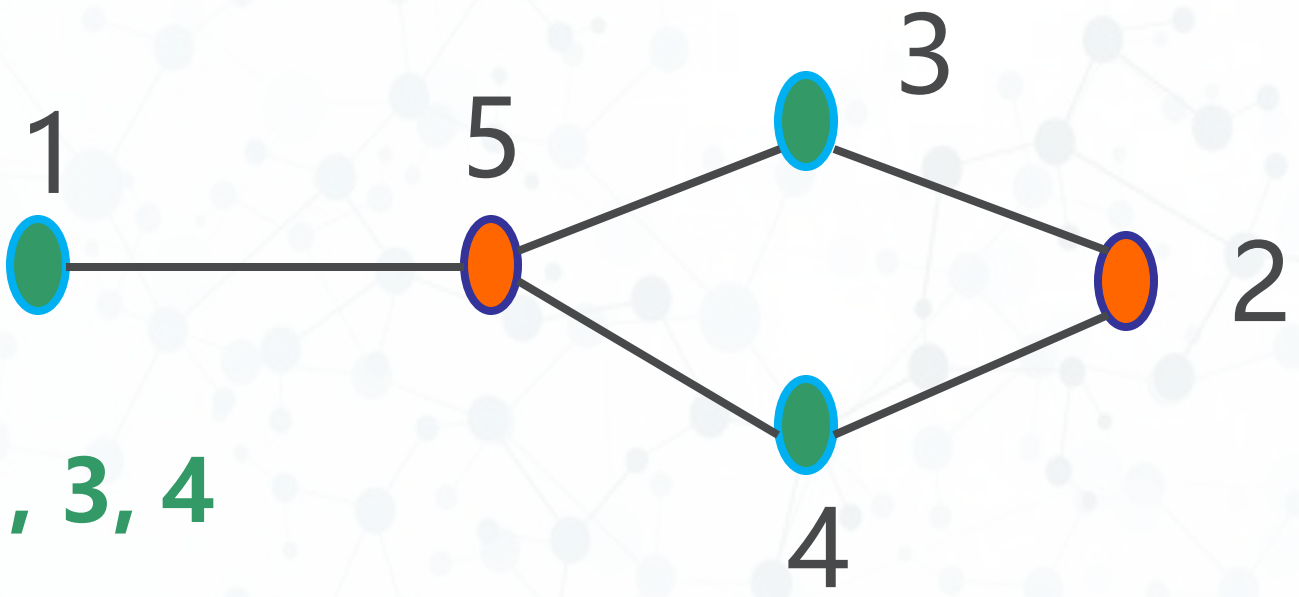
1，2

3，4

5

贪心哪！

最优解



1, 3, 4

2, 5

思考：平面图问题求解过程

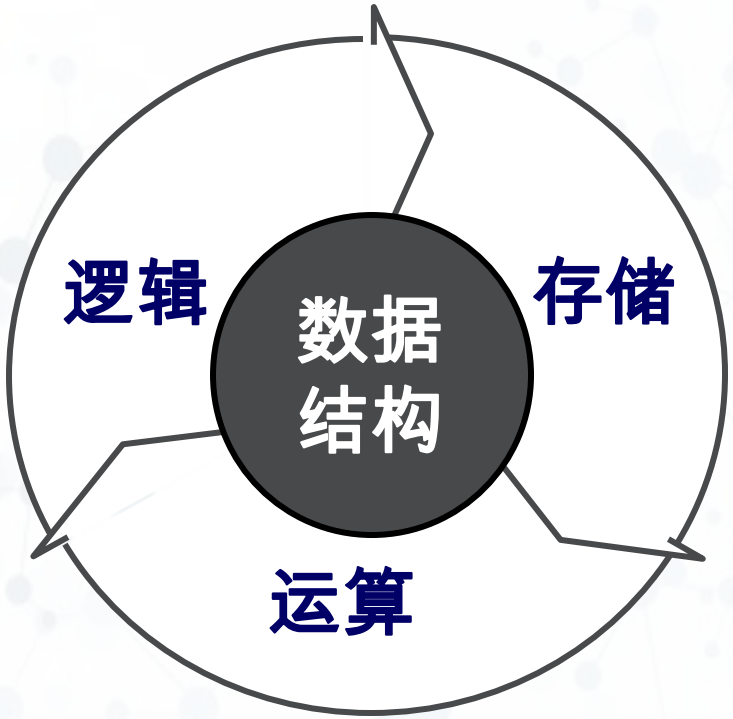
- 问题抽象？
- 数据抽象？
- 算法抽象？
- 不妨编程序模拟实现

第1章　概论

- 问题求解
- **数据结构及抽象数据类型**
- 算法的特性及分类
- 算法的效率度量

1.2 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



基本数据类型

- 整数类型(integer)
- 实数类型(real)
- 布尔类型(boolean)
  - C++语言中0表示false，非0表示true
  - 也支持 false，true 保留字
- 字符类型(char)
  - ASCII用**单个字节**（最高位委为0）表示字符
  - 汉字符号需用**2个或更多字节**（每个字节的最高位bit为1）编码
  - Unicode，GB，Big5，HZ

指针类型

- 指针类型(pointer)：指向某一内存单元的**地址**
  - 32 bit机器，4个字节表示一个指针
  - 64bit的机器， 8指针
- 指针操作
  - 分配地址
  - 赋值（另一个指针的地址值，NULL空值）
  - 比较两个指针地址
  - 指针增减一个整数量

复合数据类型

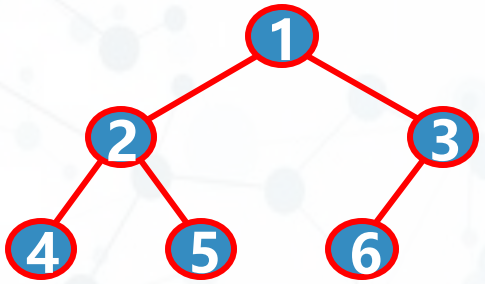
- 基本数据类型/复合类型组成的复杂结构

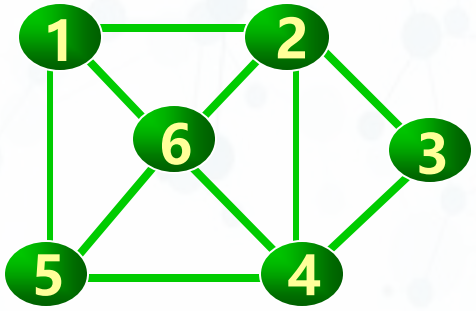
例如，

- 数组： int A[100];
- 结构： typedef struct {} B;
- 类： class C { };

数据结构的逻辑组织

- 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 ⊇ 树 ⊇ 二叉树 ⊇ 线性表



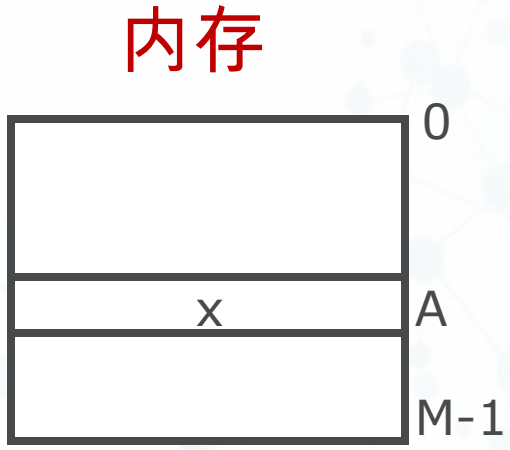


数据的存储结构

- 逻辑结构到物理存储空间的**映射**

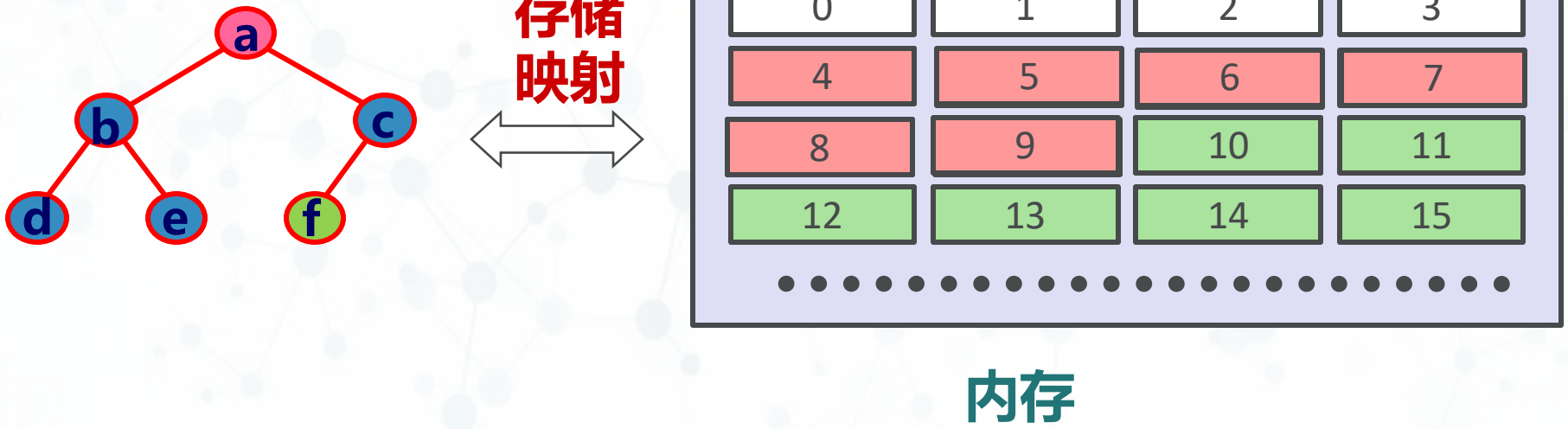
计算机主存储器（内存）

- **非负整数**地址编码，**相邻单元**的集合
  - 基本单位是字节
  - **按地址随机访问**
  - 访问不同地址所需时间基本相同



数据的存储结构

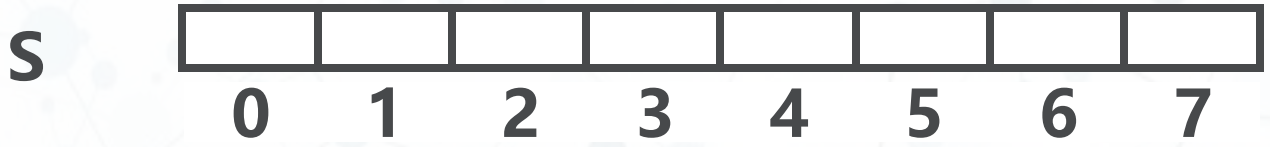
- 对逻辑结构（K，r），其中r∈R
  - 对结点集 K 建立一个从 K 到存储器 M 的单元的映射：K→M，对于每一个结点 j∈K 都对应一个**唯一**的**连续存储**区域c ∈ M



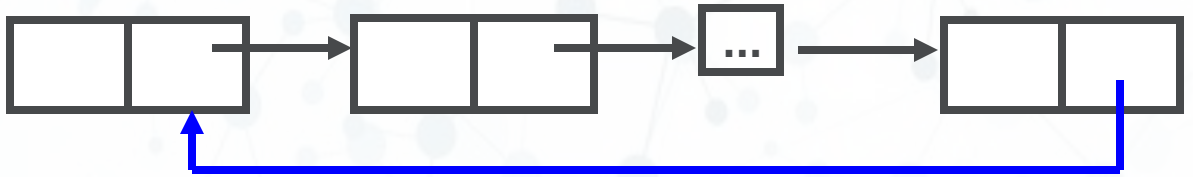
北京大学 陈斌 gischen@pku.edu.cn 2024

数据的存储结构

- 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

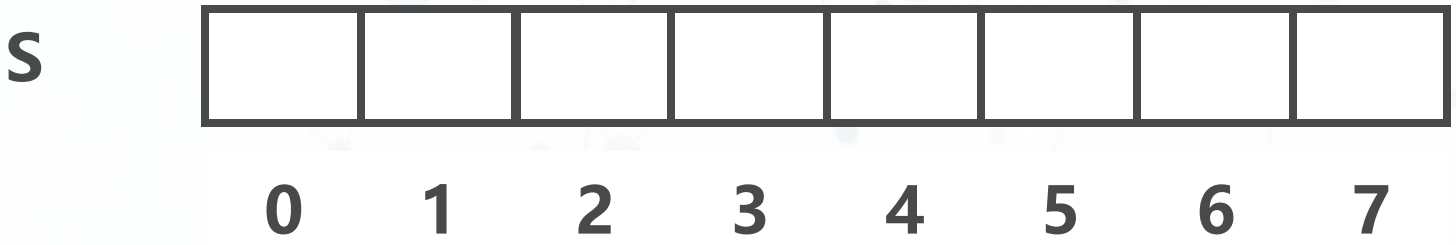


- 四类：**顺序、链接、索引、散列**

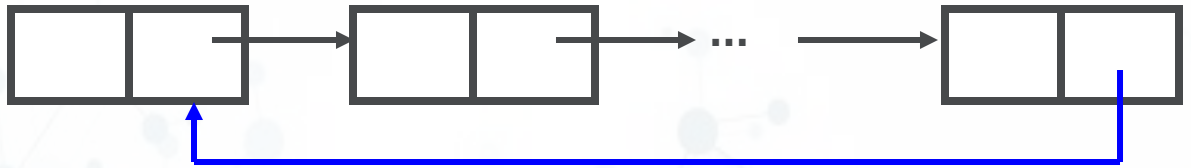
北京大学 陈斌 gischen@pku.edu.cn 2024

顺序方法

- 把一组结点存储在按**地址相邻的顺序存储**单元里
- **存储单元的顺序**，表达了结点间的逻辑后继关系
- 数组（向量）：**按下标访问**



链接方法



- 链接法： 结点的存储结构中附加指针
  - **数据域 E、指针域 P**
  - **指针的指向**，表达了两个结点的逻辑后继关系

- 存储密度($\rho \leq 1$) 数据本身占整个结构存储比率

例如，链表的存储密度

$\rho = n \times E / n(P+E) = E/(P+E)$

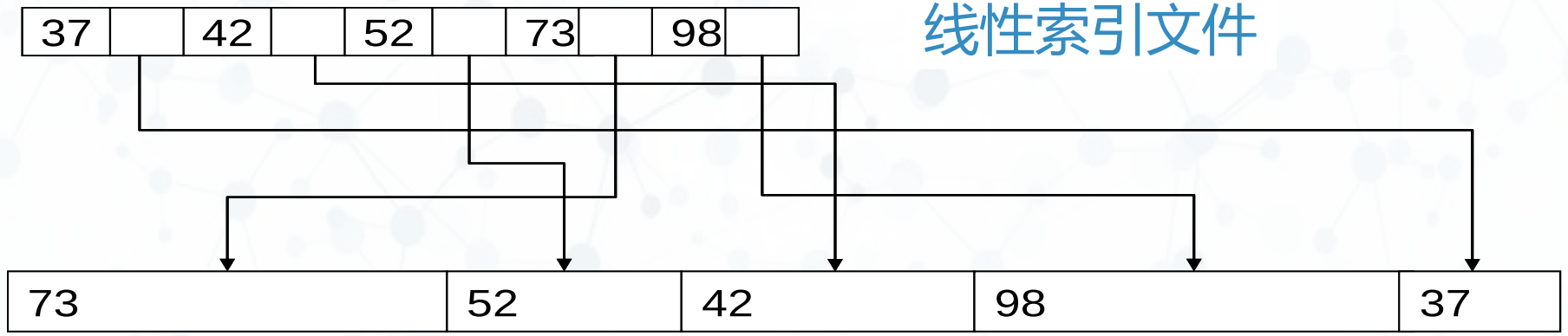
(n表示线性表中当前元素的数目)

北京大学 陈斌 gischen@pku.edu.cn 2024

索引方法

- 索引映射函数Y：Z→D
  - 结点的**索引值** z∈Z
  - 结点的**存储地址** d∈D

索引表 S



散列方法

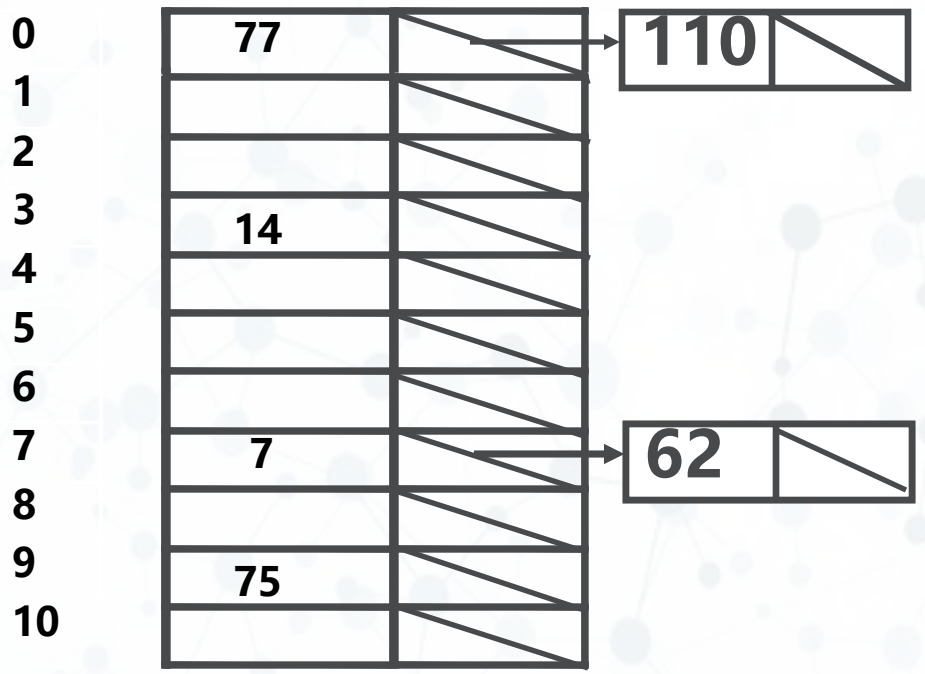
- 散列是索引方法的延伸和扩展

- **散列函数**：将关键码 s **映射** 到非负整数 z

  h: S → Z

  对任意的 s∈ S，散列函数 h(s)=z，z∈ Z

散列（hashing）的方法



$$H(k) = k \% 11$$

抽象数据类型

- 简称**ADT** (Abstract Data Type)
  - 定义了一组运算的数学模型
  - 与物理存储结构无关
  - 使软件系统建立在数据之上(面向对象)

- **模块化**的思想的发展
  - 隐藏运算实现的细节和内部数据结构
  - 软件复用

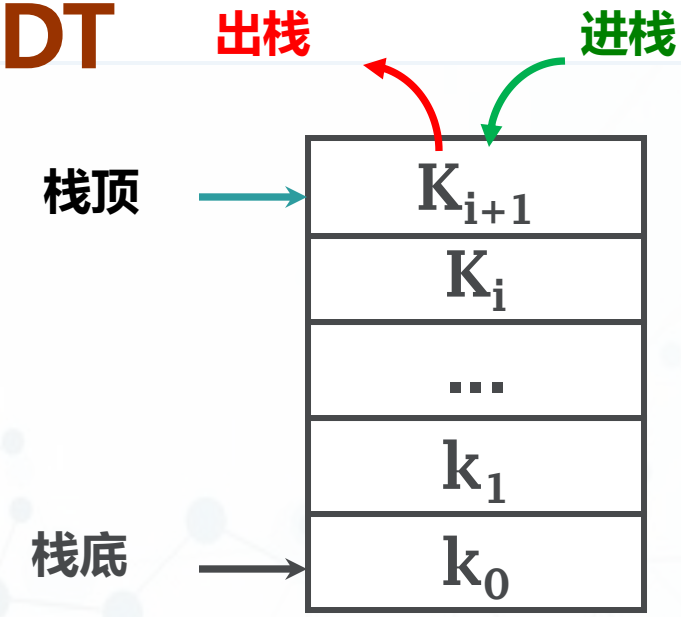
抽象数据类型ADT

- 抽象数据结构三元组

  <数据对象D，数据关系S，数据操作P>

- 先定义逻辑结构，再定义运算
  - **逻辑结构**：数据对象 + 数据关系
  - **运算**：数据操作

例：栈的抽象数据类型ADT



- 逻辑结构：线性表
- 操作特点：限制访问端口
  - 只允许在一端进行插入、删除操作
  - 入栈（push）、出栈（pop）、取栈顶（top）
    判栈空（isEmpty）

```
template <class T>                  // 栈的元素类型为 T
class Stack {
public:                             // 栈的运算集
    void clear();                   // 变为空栈
    bool push(const T item);        // item入栈，成功返回真，否则假
    bool pop(T & item);             // 弹栈顶，成功返回真，否则返回假
    bool top(T& item);              // 读栈顶但不弹出，成功真，否则假
    bool isEmpty(;                  // 若栈已空返回真
    bool isFull();                  // 若栈已满返回真
};
```

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?

第1章　概论

- 问题求解
- 数据结构及抽象数据类型
- **算法的特性及分类**
- **算法的效率度量**

问题 —— 算法 —— 程序

目标：问题求解

- **问题（problem）**一个函数
  - 从输入到输出的一种映射
- **算法（algorithm）**一种方法
  - 对特定问题求解过程的描述，是指令的有限序列
- **程序（program）**
  - 是算法在计算机程序设计语言中的实现

算法的特性

- 通用性
  - 对参数化输入进行问题求解
  - 保证计算结果的正确性
- 有效性
  - 算法是有限条指令组成的指令序列
  - 即由一系列具体步骤组成
- 确定性
  - 算法描述中的下一步应执行的步骤必须明确
- 有穷性
  - 算法的执行必须在有限步内结束
  - 换句话说，算法不能含有死循环

算法的渐进分析

$$f(n) = n^2 + 100n + \log_{10} n + 1000$$

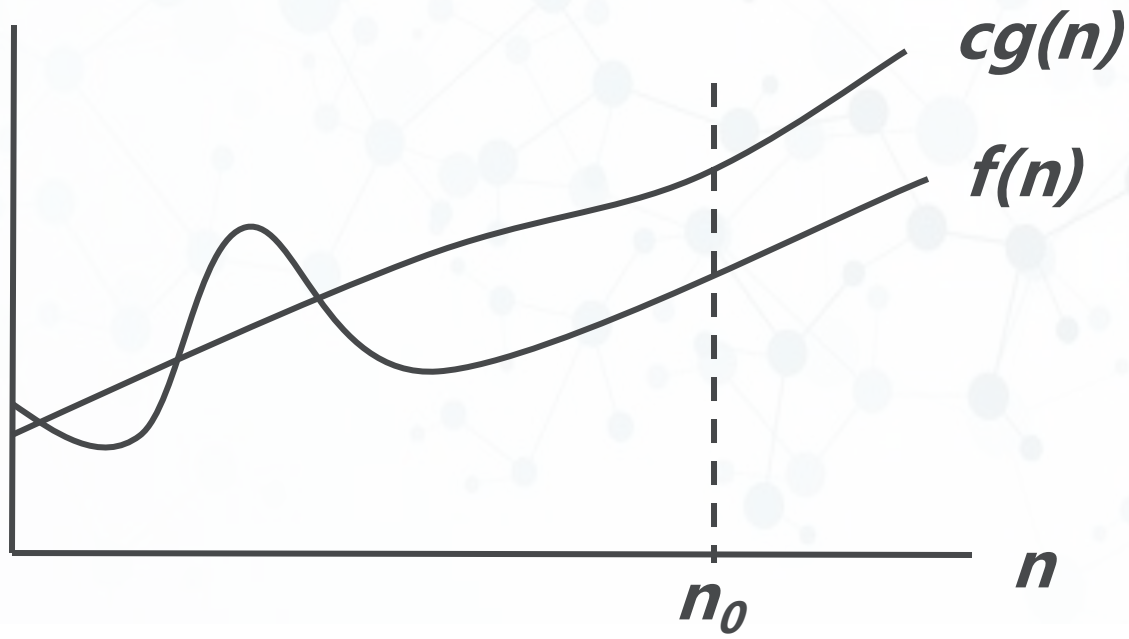
- 数据规模 n 逐步增大时，
  f(n) 的增长趋势
- 当 n 增大到一定值以后，计算公式中影响最大的就是 n 的幂次最高的项
  - 其他的常数项和低幂次项都可以忽略

算法渐进分析：大O表示法

- 函数f，g定义域为自然数，值域非负实数集
- **定义**：如果存在正数c和n_0，使得对任意的$n \geq n_0$，都有$f(n) \leq cg(n)$
- 称f(n)在集合O(g(n))中，简称f(n)是O(g(n))的，
  或 f(n) = O(g(n))
- 大O表示法：表达函数增长率上限
  - 一个函数增长率的上限可能不止一个
  - 采用大O表示法时，最好给出所有上限中**最“紧”**（也即最小）的那个上限

大 O 表示法

- f(n) = O(g(n))，当且仅当
  — 存在两个参数 c > 0，n_0 > 0, 对于所有的 n ≥ n_0，都有 f(n) ≤ cg(n)
- iff ∃ c, n_0 > 0 s.t. ∀ n ≥ n_0 : 0 ≤ f(n) ≤ cg(n)



n足够大
g(n)是 f(n) 的上界

大O表示法的单位时间

- 简单布尔或算术运算
- 简单I/O
  - 指函数的输入/输出
    例如，从数组读数据等操作
  - 不包括键盘文件等I/O
- 函数返回

大O表示法的运算法则

- **加法规则:** $f_1(n)+f_2(n)=O(\max(f_1(n), f_2(n)))$
  - 顺序结构，if 结构，switch结构
- **乘法规则:** $f_1(n)\cdot f_2(n)=O(f_1(n)\cdot f_2(n))$
  - for, while, do-while结构

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

$n-i$

?

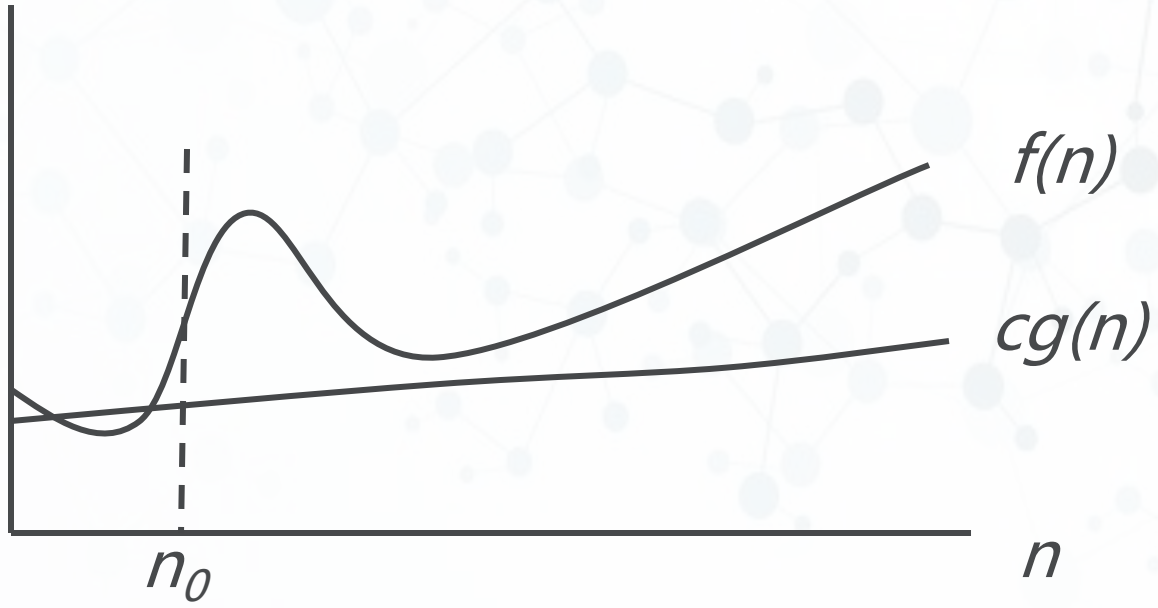
$$\sum_{i=0}^{n-1}(n-i)=\frac{n(n-1)}{2}=\frac{n^2-n}{2}=O(n^2)$$

算法渐进分析： 大Ω表示法

- **定义**： 如果存在正数c和n_0， 使得对所有的$n \geq n_0$， 都有$f(n) \geq cg(n)$
  则称f(n)在集合Ω (g(n)) 中，或简称 f(n) 是 Ω (g(n))的，
  或 f(n) = Ω (g(n))
- 大O表示法和大Ω表示法的唯一区别在于不等式的方向而已
- 大Ω表示法：表达函数增长率下限
  - 一个函数增长率的下限可能不止一个
  - 采用大Ω表示法时， 最好给出所有下限中那个最“紧”（即最大）的下限

大 Ω 表示法

- f(n) = Ω(g(n))
  - iff ∃ **c, n_0 > 0 s.t. ∀ n ≥ n0 , 0 ≤ cg(n) ≤ f(n)**
- 与大O表示法的唯一区别在于不等式的方向



n足够大
g(n)是 f(n) 的下界

算法渐进分析： 大Θ表示法

- 当上、下限相同时则可用大Θ表示法
- 定义如下：

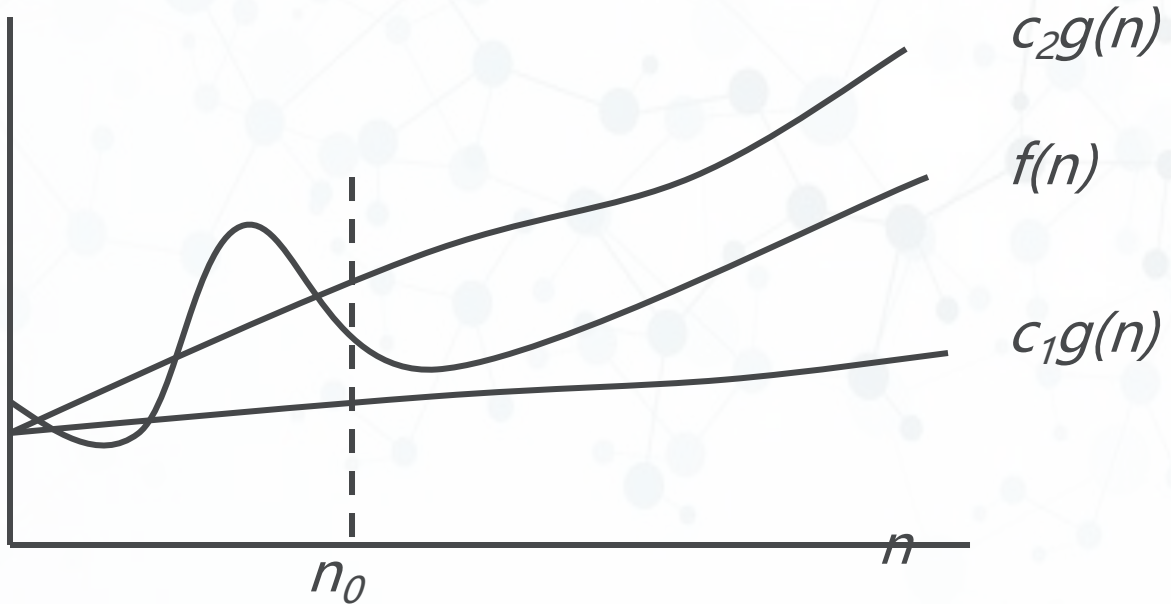
  如果一个函数既在集合O (g(n))中又在集合Ω (g(n))中，则称其为Θ (g(n))。

- 也即，当上、下限相同时则可用大Θ表示法
- 存在正常数c_1, c_2，以及正整数n_0，使得对于任意的正整数$n > n_0$，有下列两不等式同时成立：

$$c_1\, g(n) \le f(n) \le c_2\, g(n)$$

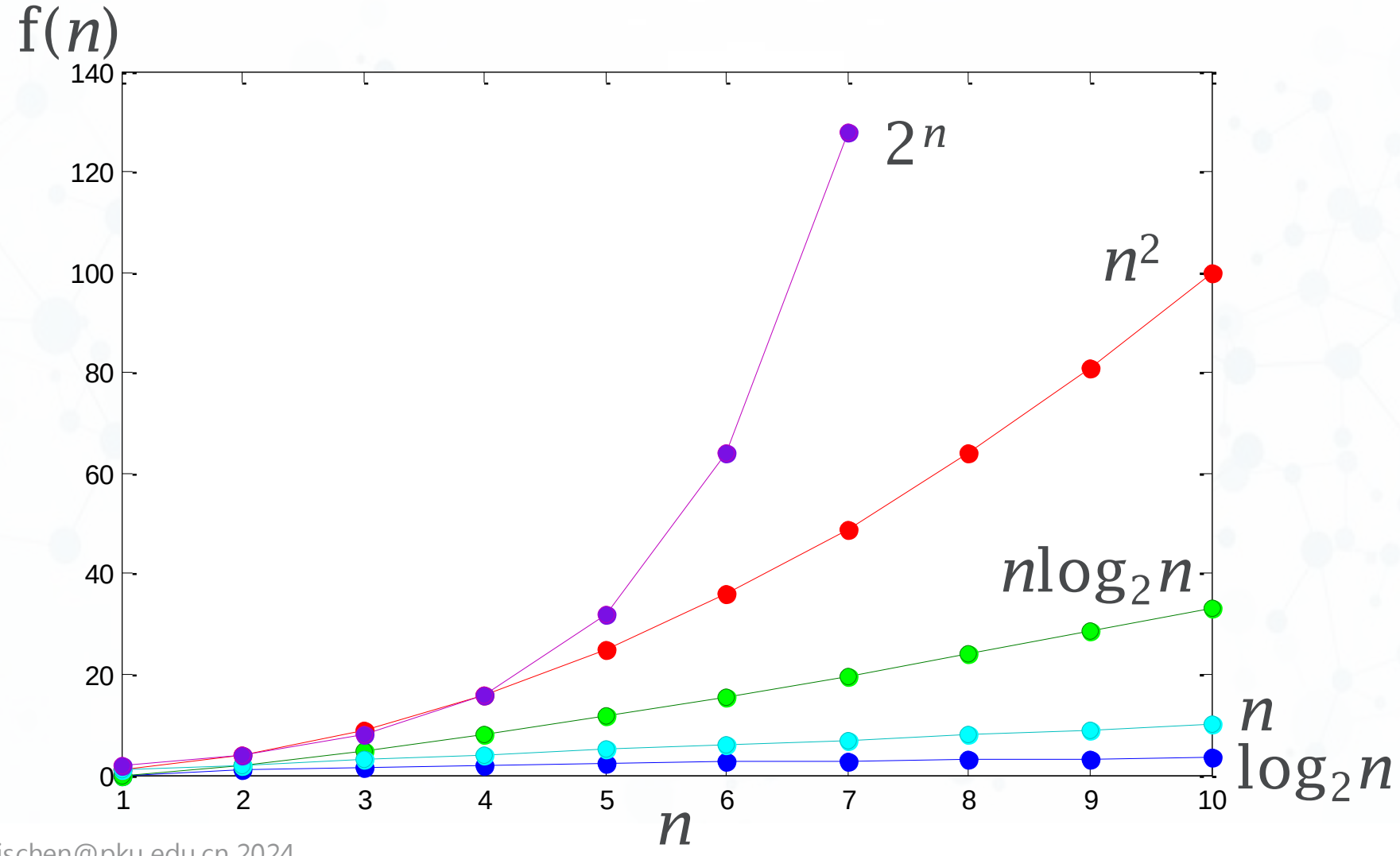
大 Θ 表示法

- f(n) = Θ(g(n))
  - iff ∃ c_1, c_2, n_0 > 0 s.t. $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$, ∀ n >= n_0
- 上、下限相同，则可用大 Θ 表示法



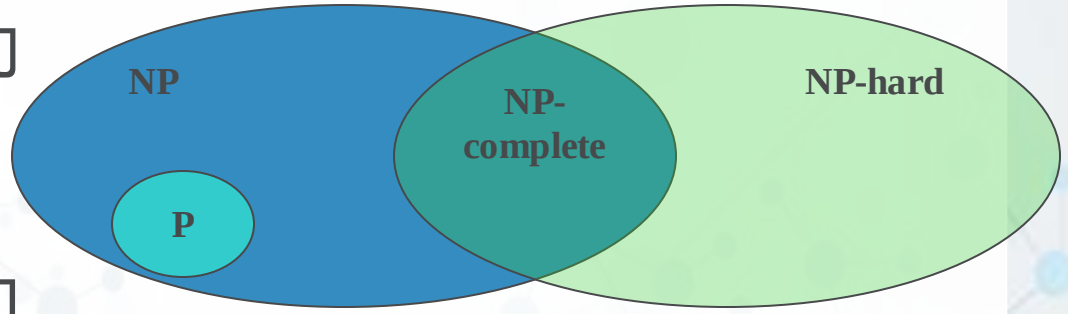
n足够大
f(n) 与 g(n) 增长率一样

增长率函数曲线

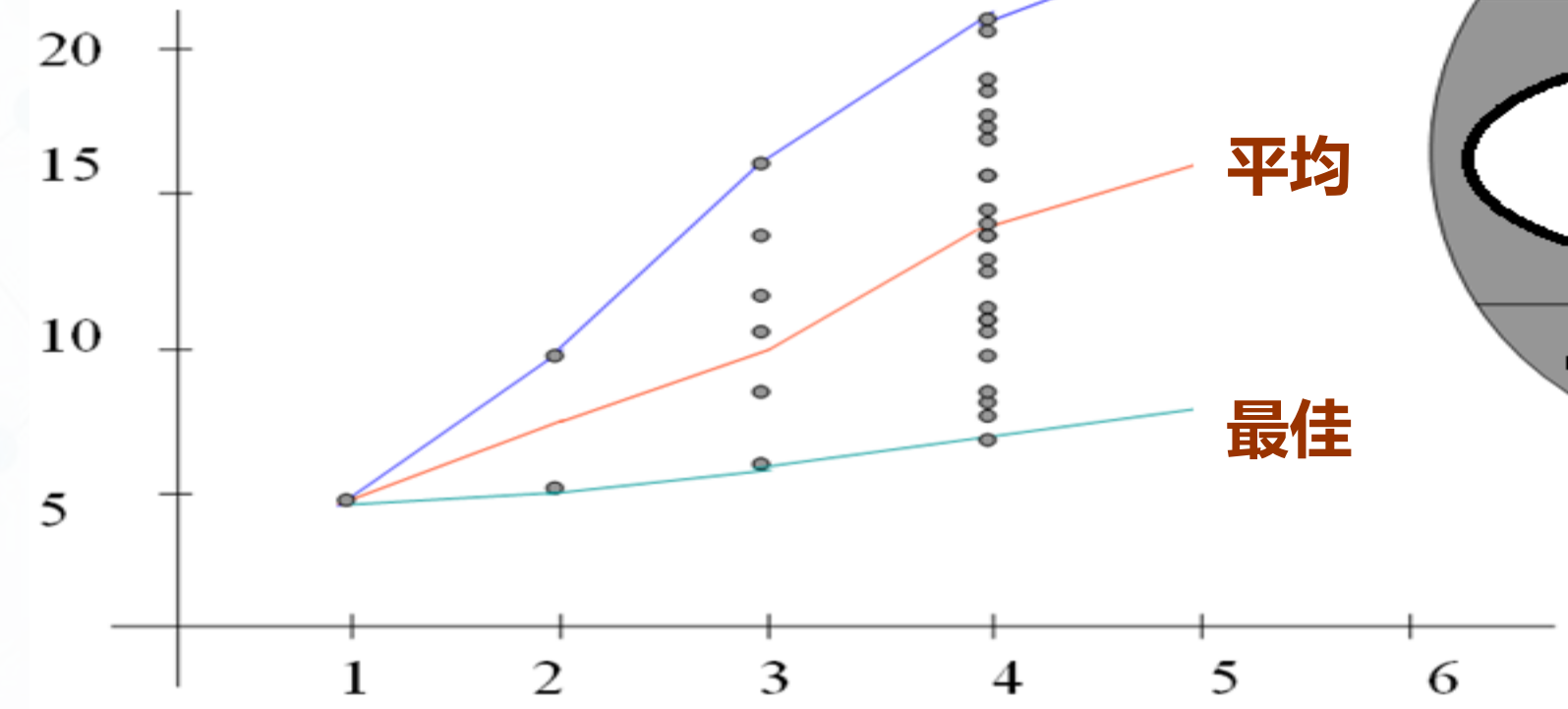


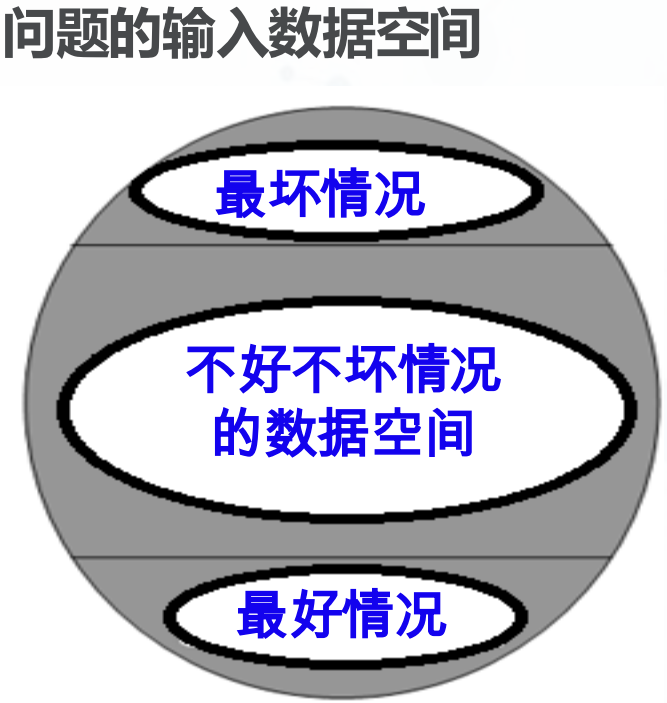
计算复杂性理论

- 不可解问题：虽然能够被准确定义，但却不存在能够解决该问题的算法
  - 停机问题
- 难解问题：求解算法均无法在多项式时间 n^k 数量级内解决（其中 k 是任意正整数）
  - 最优巡游路径问题
  - 判定命题逻辑公式是否为恒真
  - ……
- 组合爆炸型的难解问题
  - 随着问题的规模 n 的增大，算法的时间开销不能约束在 n 的 k 阶多项式数量范围内（常数k不依赖于 n ）
  - 八皇后、河内塔



问题空间 vs 时间开销


时间代价
20
15
10
5
1
2
3
4
5
6
规模n
最坏
平均
最佳


问题的输入数据空间
最坏情况
不好不坏情况
的数据空间
最好情况

顺序找k值

- 顺序从一个规模为n的一维数组中找出一个给定的K值
- 最佳情况
  - 数组中第1个元素就是K
  - 只要检查一个元素
- 最差情况
  - K是数组的最后一个元素
  - 检查数组中所有的n个元素

顺序找k值——平均情况

- 如果等概率分布
- K值出现在n个位置上概率都是1/n则平均代价

$$\frac{1+2+\ldots+n}{n}=\frac{n+1}{2}$$

顺序找k值——平均情况

- 概率不等
  - 出现在第1个位置的概率为1/2
  - 第2个位置上的概率为1/4
  - 出现在其他位置的概率都是

$$\frac{1-1/2-1/4}{n-2}=\frac{1}{4(n-2)}$$

  - 平均代价

$$\frac{1}{2}+\frac{2}{4}+\frac{3+\ldots+n}{4(n-2)}=1+\frac{n(n+1)-6}{8(n-2)}=1+\frac{n+3}{8}$$

二分法检索

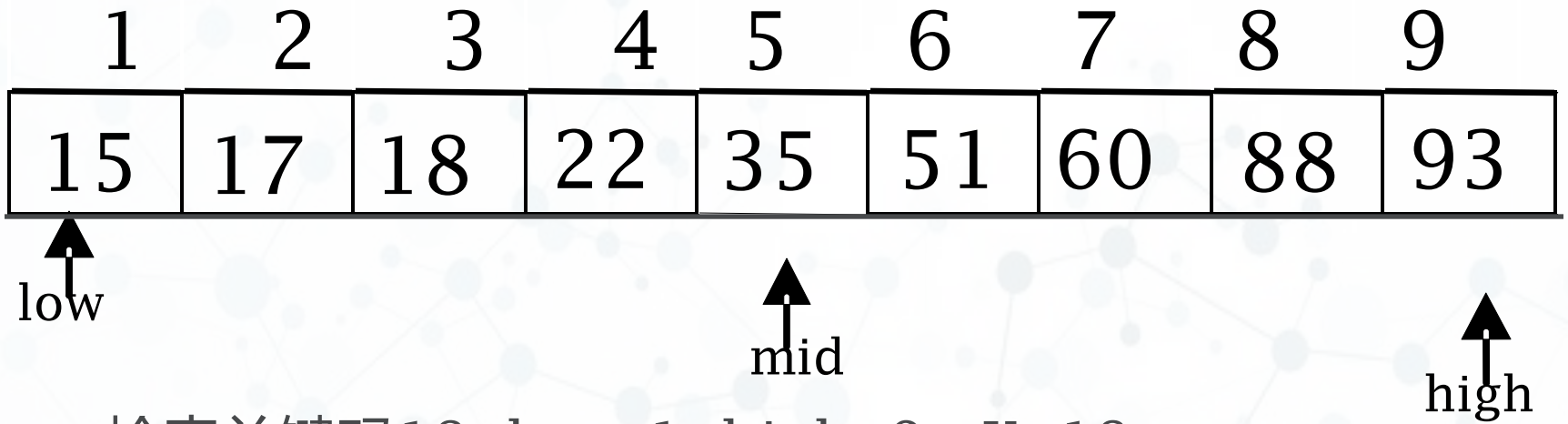
对于已排序顺序线性表

- 数组中间位置的元素值 k_{mid}
  - 如果$k_{mid} = k$，那么检索工作就完成了
  - 当$k_{mid} > k$时，检索继续在前半部分进行
  - 相反地，若$k_{mid} < k$，就可以忽略mid以前的那部分，检索继续在后半部分进行
- 快速
  - $k_{mid} = k$ 结束
  - $K_{mid} \neq k$ 起码缩小了一半的检索范围

二分法检索算法

```
template  <class Type> int BinSearch (vector<Item<Type>*>& dataList, int
length, Type k){
    int low=1, high=length, mid;
    while (low<=high)  {
        mid=(low+high)/2;
        if (k<dataList[mid]->getKey())
              high = mid-1;                  //右缩检索区间
        else if (k>dataList[mid]->getKey())
              low = mid+1;                   //左缩检索区间
        else return mid;                     //成功返回位置
     }
    return 0; //检索失败，返回0
}
```

二分法检索图示



检索关键码18 low=1 high=9 K=18

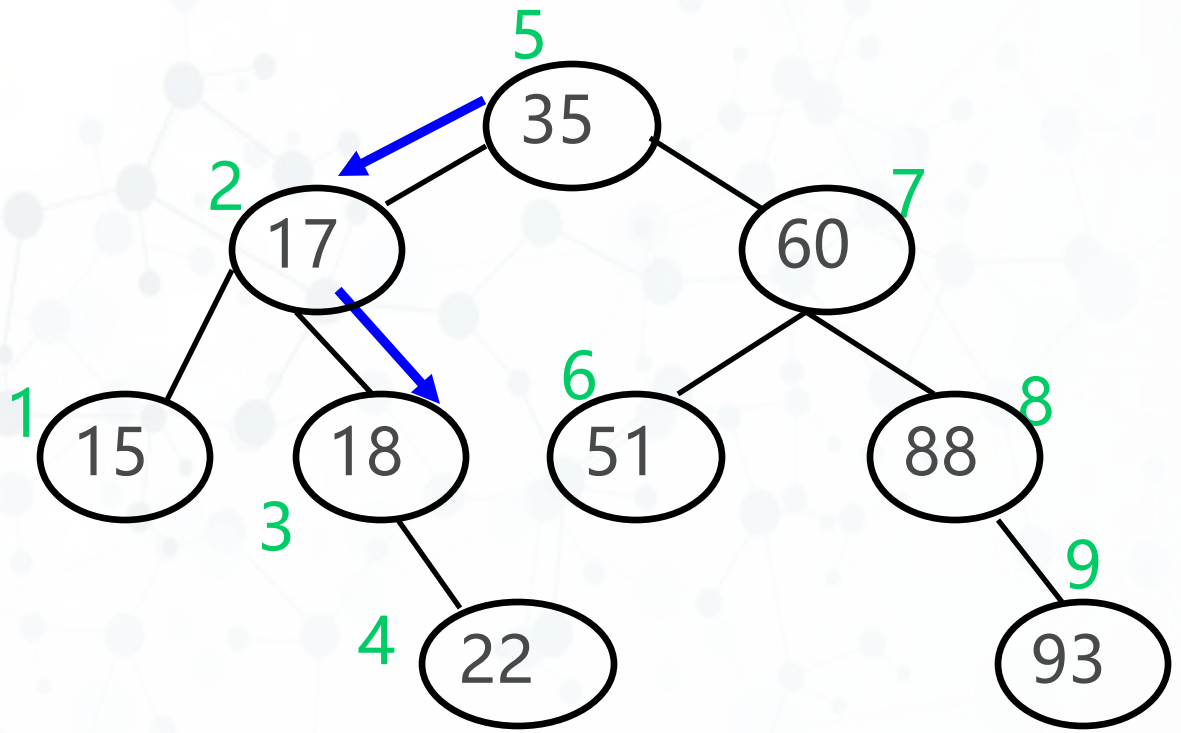
第一次：mid=5; array[5]=35>18
high=4; (low=1)
第二次：mid=2; array[2]=17<18
low=3; (high=4)
第三次：mid=3; array[3]=18=18
mid=3； return 3

二分法检索性能分析

- 最大检索长度为 $\lceil \log_2(n+1) \rceil$
- 失败的检索长度是 $\lceil \log_2(n+1) \rceil$

或 $\lfloor \log_2(n+1) \rfloor$

- 在算法复杂性分析中
  - log n是以2为底的对数
  - 以其他数值为底，算法量级不变

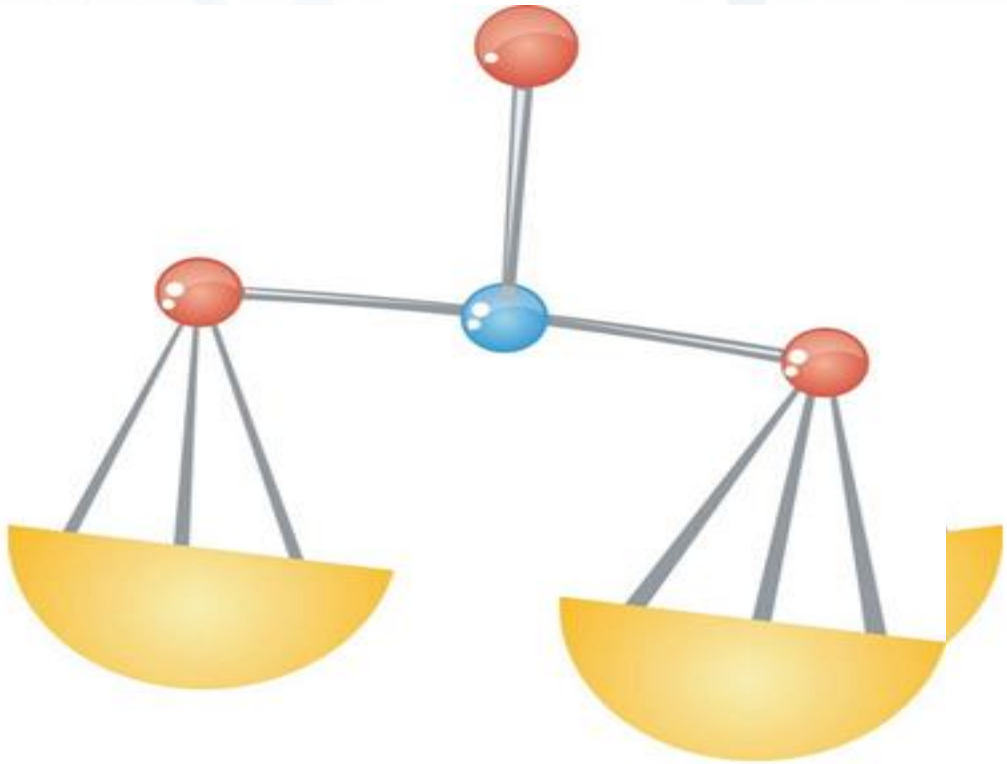


时间/空间权衡

- **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
  - 软件工程

时空权衡

- 增大空间开销可能改善算法的时间开销
- 可以节省空间，往往需要增大运算时间



数据结构和算法的选择

- 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
    ——— 问题抽象、数据抽象
  - 数据结构的初步设计往往先于算法设计
- 注意数据结构的可扩展性
  - 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求解问题的演变和扩展

思考：数据结构和算法的选择

- 算法设计目标？
- 算法选择的过程?