第9讲 算法：程序与计算系统之灵魂

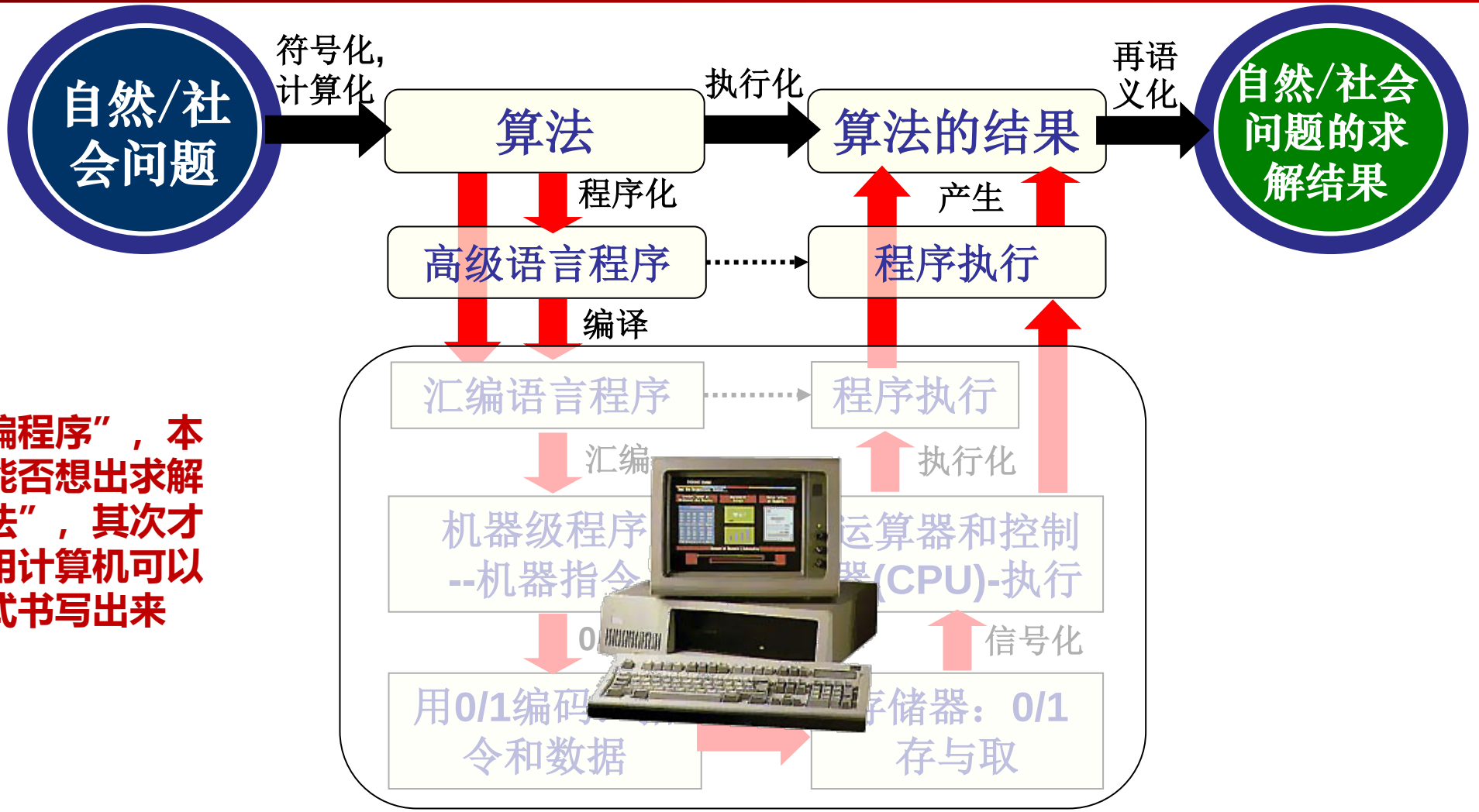
李玉华

华中科技大学计算机学院

idcliyuhua@hust.edu.cn

第9讲 算法：程序与计算系统之灵魂

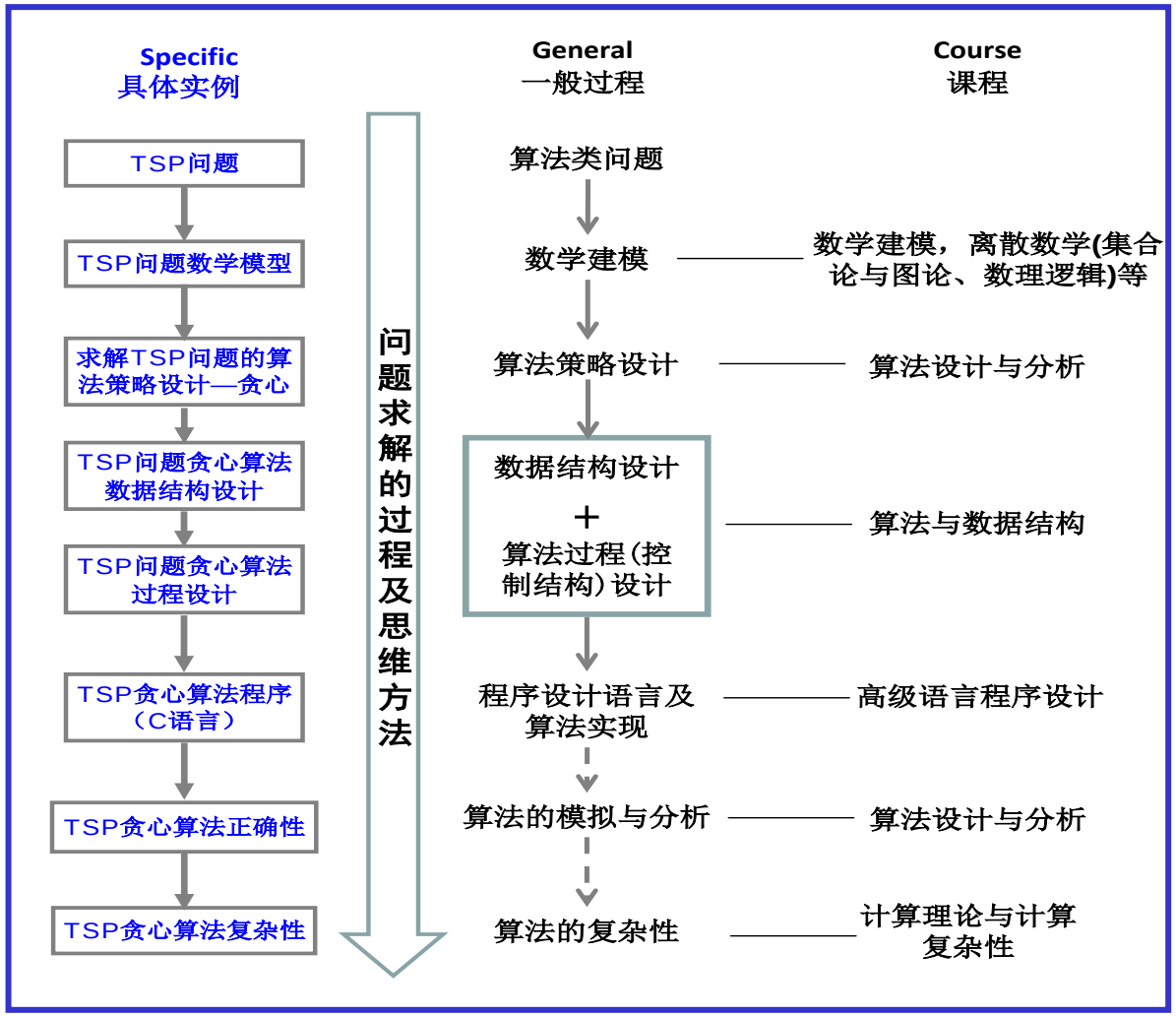
概述



“是否会编程序”，本质上是“能否想出求解问题的算法”，其次才是将算法用计算机可以识别的形式书写出来

第9讲 算法：程序与计算系统之灵魂

本讲学习什么--理解算法类问题求解框架



第9讲 算法：程序与计算系统之灵魂

一、算法的概念与特征

二、问题与数学建模

三、算法策略选择（算法思想）

四、算法的数据结构及其操作

五、算法的控制结构及其表达方法

六、算法的程序设计

七、算法分析

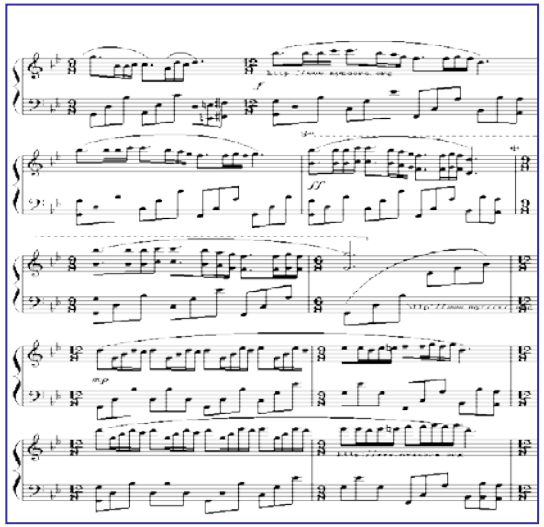
TSP问题
求解为例

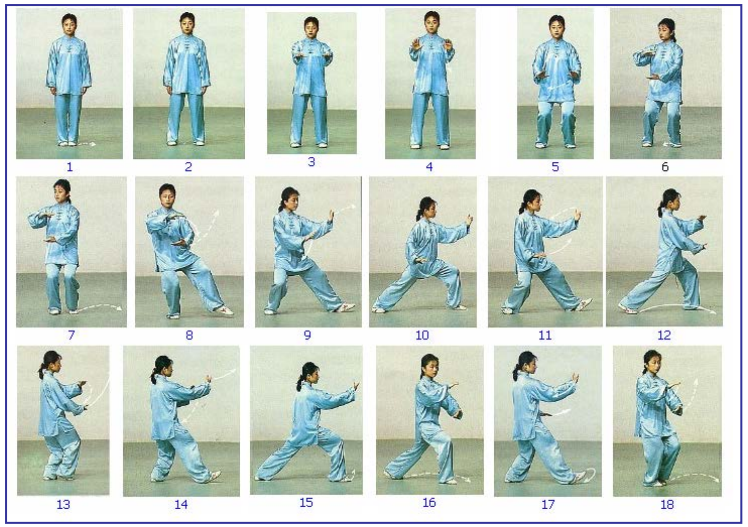
算法的概念与特征

什么是算法?

算法---计算机与软件的灵魂。“算法”(Algorithm)一词源于阿拉伯数学家阿科瓦里茨米(AlKhowarizmi)，其在公元825年写了著名的《波斯教科书》(Persian Textbook)，书中概括了进行四则算术运算的计算规则。

算法是一个有穷规则的集合，它用规则规定了解决某一特定类型问题的运算序列，或者规定了任务执行或问题求解的一系列步骤。





如音乐乐谱、太极拳谱等都可看作广义的算法

算法的概念与特征

算法示例

寻找两个正整数的最大公约数的欧几里德算法

输入：正整数*M*和正整数*N*

输出：*M*和*N*的最大公约数(设M>N)

算法步骤：

Step 1. *M*除以*N*,记余数为*R*

Step 2. 如果*R*不是*0*，将*N*的值赋给*M*, *R*的值赋给*N*, 返回Step 1; 否则, 最大公约数是*N*, 输出*N*, 算法结束。

	M	N	R	最大公约数
具体问题	32	24		?
算法计算过程				
1	32	24	8	
2	24	8	0	8
具体问题	31	11		?
算法计算过程				
1	31	11	9	
2	11	9	2	
3	9	2	1	
4	2	1	0	1

算法的概念与特征

算法的五个特征

- **有穷性**：一个算法在执行有穷步规则之后必须结束。
- **确定性**：算法的每一个步骤必须要确切地定义，不得有歧义性。
- **输入**：算法有零个或多个的输入。
- **输出**：算法有一个或多个的输出/结果，即与输入有某个特定关系的量。
- **能行性**：算法中有待执行的运算和操作必须是相当基本的**(可以由机器自动完成)**，并能在有限时间内完成。

寻找两个正整数的最大公约数的欧几里德算法

输入：正整数*M*和正整数*N*

输出：*M*和*N*的最大公约数(设M>N)

算法步骤：

Step 1. *M*除以*N*,记余数为*R*

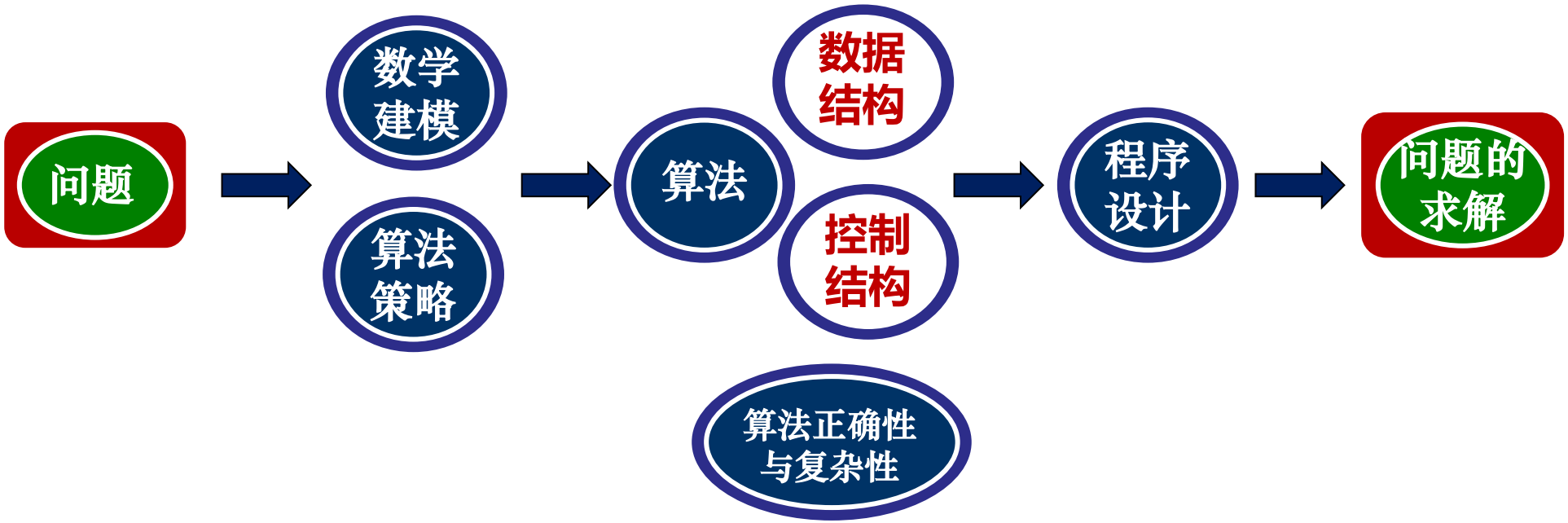
Step 2. 如果*R*不是*0*，将*N*的值赋给*M*, *R*的值赋给*N*, 返回Step 1; 否则, 最大公约数是*N*, 输出*N*, 算法结束。

基本运算：除法、赋值、逻辑判断

典型的“重复/循环”与“迭代”

算法的概念与特征

算法相关的知识



第9讲 算法：程序与计算系统之灵魂

一、算法的概念与特征

二、问题与数学建模

三、算法策略选择（算法思想）

四、算法的数据结构及其操作

五、算法的控制结构及其表达方法

六、算法的程序设计

七、算法分析

TSP问题
求解为例

问题与数学建模

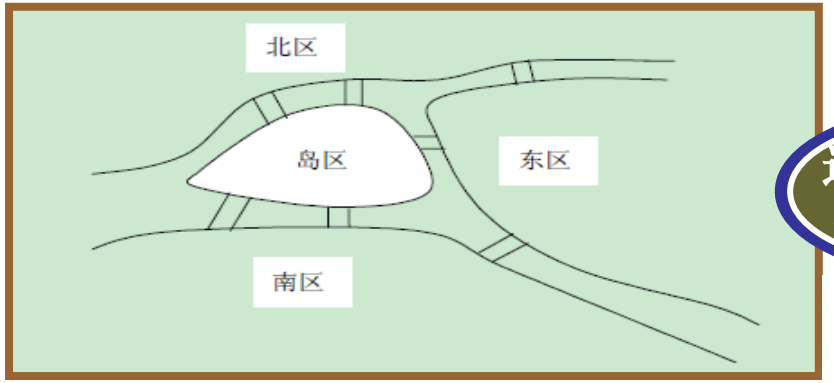
算法类问题求解的第一步就是数学建模

数学建模是用数学语言描述实际现象的过程，即建立数学模型的过程。

数学模型是对实际问题的一种数学表述，是关于部分现实世界为某种目的的一个抽象的简化的数学结构。

【**示例**】哥尼斯堡七桥问题：

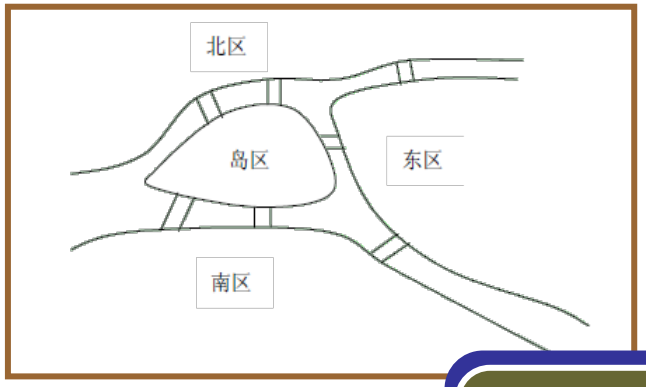
“寻找走遍这7座桥且只许走过每座桥一次最后又回到原出发点的路径”。



问题与数学建模

将社会/自然问题 转化为 数学问题

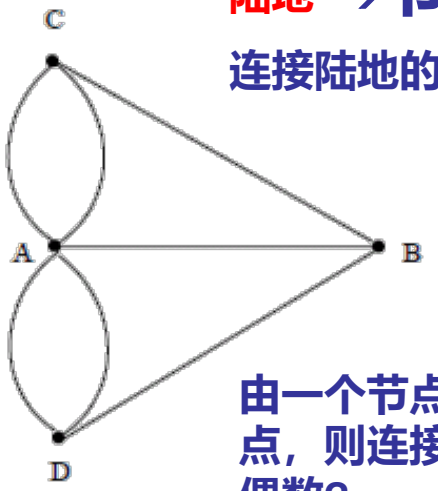
哥尼斯堡七桥问题： 抽象成一个“图(Graph)”,由节点和边所构成的一种结构。



这个问题无解！
无解的问题还
用构造算法吗？

陆地--→节点

连接陆地的桥梁--→连接节点的边



由一个节点出发，又回到原出发节点，则连接该节点的边数是否应为偶数？

问题与数学建模

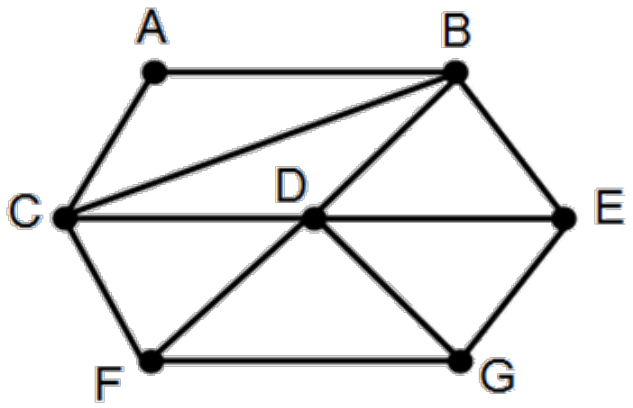
数学模型，可描述一般性问题

哥尼斯堡七桥问题:

“寻找走遍这7座桥且只许走过每座桥一次最后又回到原出发点的路径”

一般性问题:

“对给定的任意一个河道图与任意多座桥判定可能不可能每座桥恰好走过一次?”。



如能抽象成数学模型，则可将一个具体问题的求解，推广为一类问题的求解！

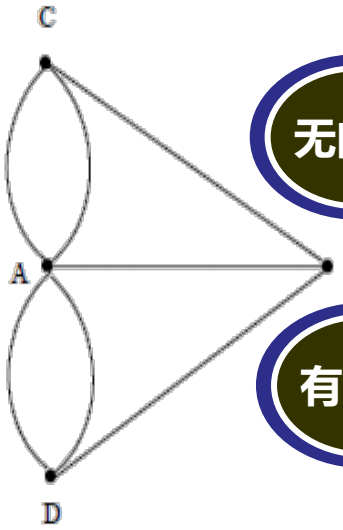
问题与数学建模

数学模型，可发现不同类别的问题及其性质

“**连通**----两个节点间有路径相连接”
“**可达**----从一个节点出发能够到达另一个节点”
“**回路**---从一个节点出发最后又回到该节点的一条路径”

经过图中每边一次且仅一次的回路

欧拉回路-欧拉图



无向图

节点的“度”

偶度节点

奇度节点

有向图

入度/出度

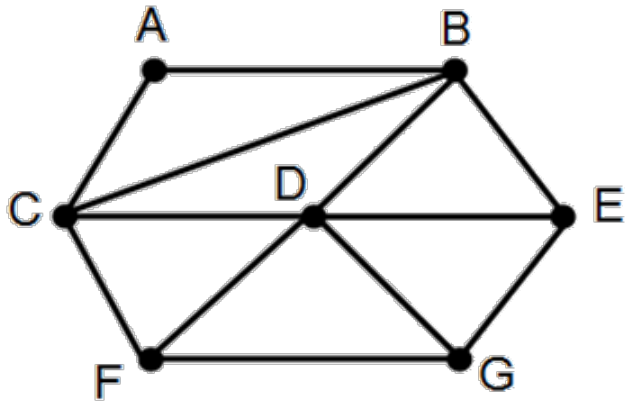
哈密尔顿回路-哈密尔顿图

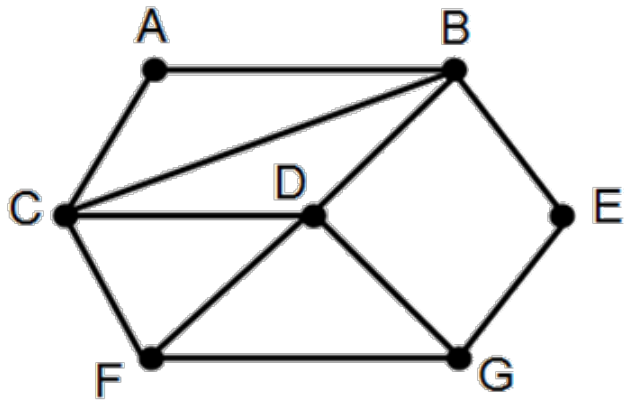
经过图中每节点一次且仅一次的回路

“图论”专门讲这方面的内容哟！

问题与数学建模

数学模型，可发现问题求解的基本思路





“一笔画”

- 凡是由偶度点组成的连通图，一定可以一笔画成。画时可以把任一偶度点为起点，最后一定能以这个点为终点画完此图；
- 凡是只有两个奇度点的连通图，其余都为偶度点，一定可以一笔画成。画时必须以一个奇度点为起点，另一个奇度点为终点。
- 其余情况都不能一笔画出。

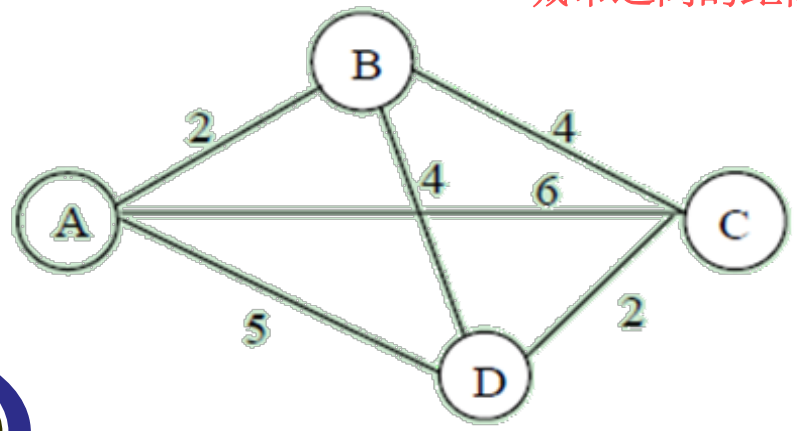
问题与数学建模【示例】TSP问题

什么是TSP问题

◆TSP问题(**Traveling Salesman Problem**，旅行商问题)，威廉哈密尔顿爵士和英国数学家克克曼T.P.Kirkman于19世纪初提出TSP问题.

◆**TSP问题**：有若干个城市，任何两个城市之间的距离都是确定的，现要求一旅行商从某城市出发必须经过每一个城市且只能在每个城市逗留一次，最后回到原出发城市，问如何事先确定好一条最短的路线使其旅行的费用最少。

城市之间的距离



经过图中每节点一次且仅一次的回路

哈密尔顿回路-哈密尔顿图

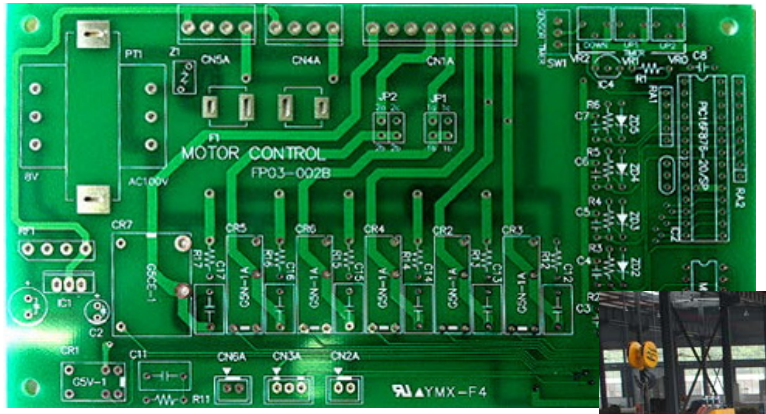
赋权-哈密尔顿图

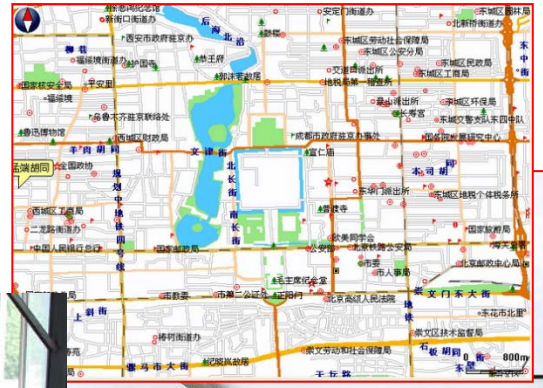
经过图中每节点一次且仅一次的回路；连接节点的边有“权值”

问题与数学建模【示例】TSP问题

现实中的很多问题都是TSP问题

◆TSP是最有代表性的**组合优化**问题之一，在半导体制造(线路规划)、物流运输(路径规划)等行业有着广泛的应用。





问题与数学建模【示例】TSP问题

将TSP问题抽象为数学问题

输入： n个城市，记为V={v_1,v_2, …, v_n}，任意两个城市$v_i, v_j \in V$之间有距离$d_{v_i v_j}$

输出： 所有城市的一个访问顺序T={t_1,t_2, …, t_n}，其中$t_i \in V$，$t_{n+1} = t_1$，使得$min \sum_{i=1}^{n} d_{t_i t_{i+1}}$.

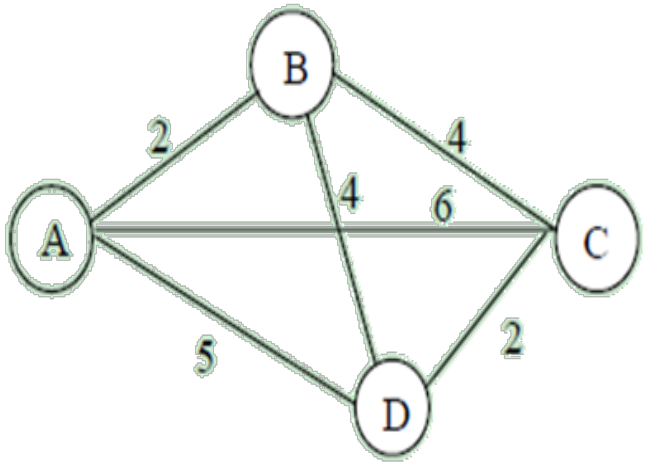
进一步简化

输入： n个城市，记为V={1, 2, …, n}，任意两个城市i，j之间的距离有$d_{i,j}$

输出： 所有城市的一个访问顺序 T={t_1,t_2, …, t_n}，其中$t_i \in V$，$t_{n+1} = t_1$,使得$\min \sum_{i=1}^{n} d_{t_i t_{i+1}}$.

问题求解的基本思想： 在所有可能的访问顺序T构成的状态空间Ω上搜索使得$\sum_{i=1}^{n} d_{t_i t_{i+1}}$最小的访问顺序$T_{opt}$。

蛮干/遍历



问题与数学建模【示例】TSP问题

现实问题抽象后是否是相同的问题?

	物流配送中的路径规划	电路板上机器钻孔的路径规划	旅行线路的路径规划 (TSP 问题)	数学抽象 (组合优化问题)
问题简要表述	有 n 个地点需要送货，怎样一个次序，才能使送货距离最短	有 n 个位置需要钻孔，怎样一个次序，才能使钻头移动距离最短	有 n 个城市需要旅行，怎样一个次序，才能使旅行者花费的交通成本最低	有 n 个节点需要访问，怎样一个次序，才能使访问者访问路径的“权值和”最短
“输入/已知”的抽象表示	n 个地点 $V_1,\ldots,V_n$，可简化为用编号 1,…,n 表示	n 个位置 $V_1,\ldots,V_n$，可简化为用编号 1,…,n 表示	n 个城市 $V_1,\ldots,V_n$，可简化为用编号 1,…,n 表示	n 个节点 $V_1,\ldots,V_n$，可简化为用编号 1,…,n 表示
“输入/已知”的抽象表示	任何两个地点 i 和 j 的距离表示为 $d_{i,j}$	任何两个位置 i 和 j 的距离表示为 $d_{i,j}$	任何两个城市 i 和 j 的交通成本表示为 $d_{i,j}$	任何两个节点 i 和 j 的权值表示为 $d_{i,j}$
“输出/结果”的抽象表示	求一路径 T，$T=(T_1,T_2,\ldots,T_n)$，其中 T_i 为 1…n 中的某一个	求一路径 T，$T=(T_1,T_2,\ldots,T_n)$，其中 T_i 为 1…n 中的某一个	求一路径 T，$T=(T_1,T_2,\ldots,T_n)$，其中 T_i 为 1…n 中的某一个	求一路径 T，$T=(T_1,T_2,\ldots,T_n)$，其中 T_i 为 1…n 中的某一个
“输出/结果”需满足约束	任意两个不同的 i,j，则 $T_i \neq T_j$	任意两个不同的 i,j，则 $T_i \neq T_j$	任意两个不同的 i,j，则 $T_i \neq T_j$	任意两个不同的 i,j，则 $T_i \neq T_j$
“输出/结果”需优化达到的目标	T 要使得 $\sum_{i=1}^{n} d_{T_i T_{i+1}}$ 最小，其中 $d_{T_n,T_{n+1}}$ 为最后地点与出发地点的距离。	T 要使得 $\sum_{i=1}^{n} d_{T_i T_{i+1}}$ 最小，其中 $d_{T_n,T_{n+1}}$ 为最后地点与出发地点的距离。	T 要使得 $\sum_{i=1}^{n} d_{T_i T_{i+1}}$ 最小，其中 $d_{T_n,T_{n+1}}$ 为最后城市与出发城市的距离。	T 要使得 $\sum_{i=1}^{n} d_{T_i T_{i+1}}$ 最小，其中 $d_{T_n,T_{n+1}}$ 为最后节点与第一个节点的权值。

通过自然数及其下标将不同类型的数据关联起来，是算法类问题抽象的重要方法。

表达四个核心要素：

- **输入**
- **输出**
- **约束**
- **目标**

第9讲 算法：程序与计算系统之灵魂

一、算法的概念与特征

二、问题与数学建模

三、算法策略选择（算法思想）

四、算法的数据结构及其操作

五、算法的控制结构及其表达方法

六、算法的程序设计

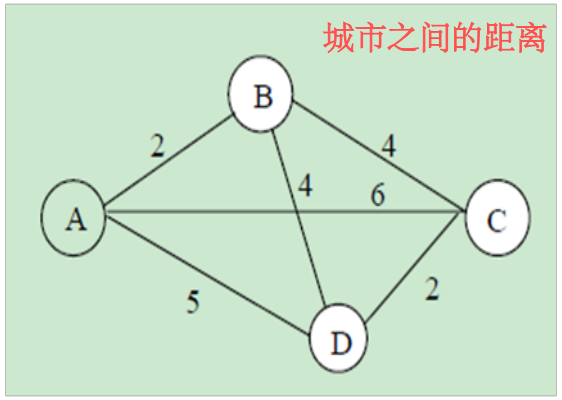
七、算法分析

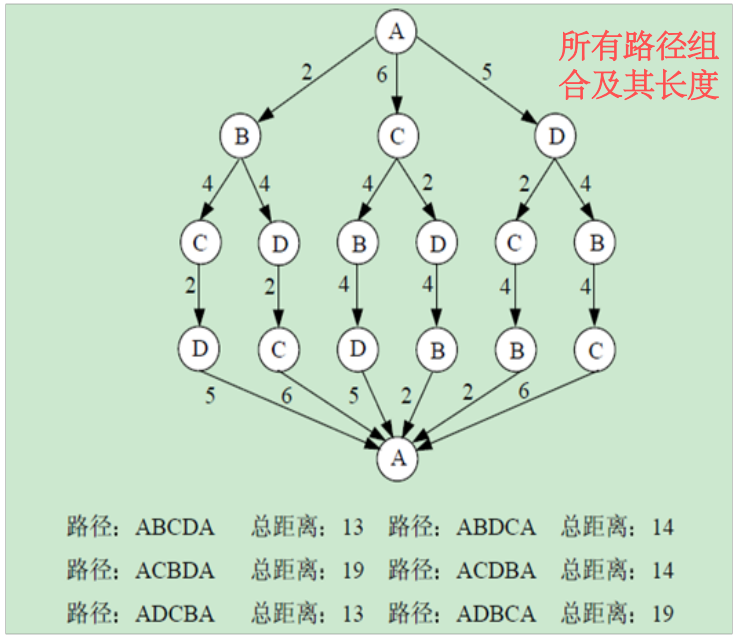
TSP问题
求解为例

三、算法策略选择【示例】TSP问题

蛮干/遍历是基本的算法策略

遍历算法求解TSP问题：列出每一条可供选择的路线，计算出每条路线的总里程，最后从中选出一条最短的路线。



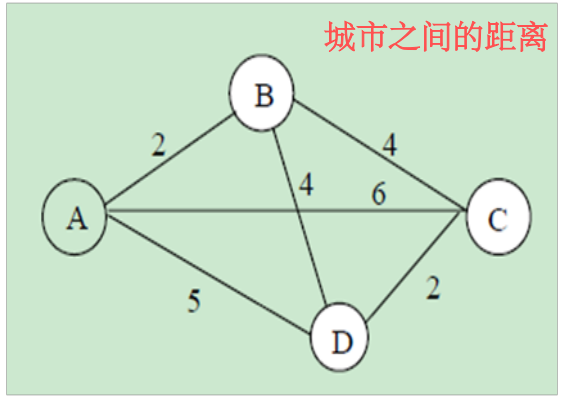


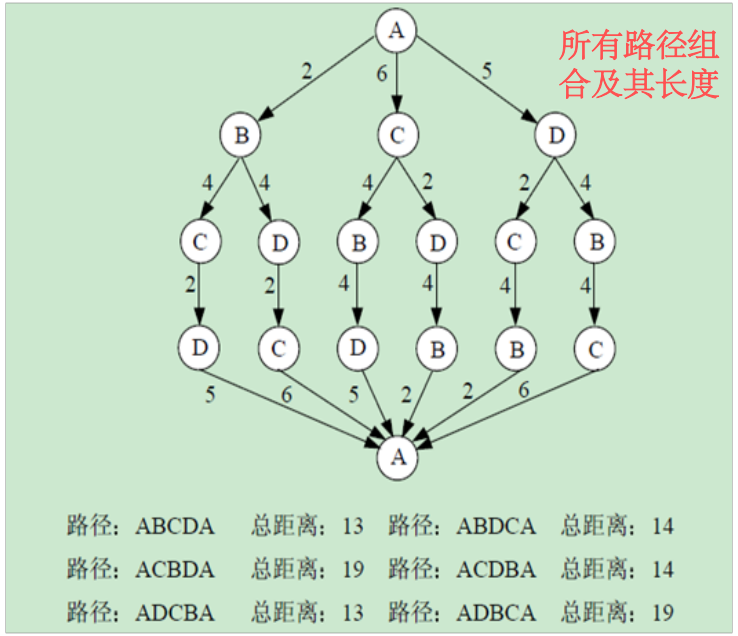
三、算法策略选择【示例】TSP问题

蛮干/遍历策略会带来怎样的问题?

组合爆炸

- ✓路径组合数目：(n-1)!
- ✓20个城市，遍历总数1.216×10^{17}
- ✓计算机以每秒检索1000万条路线的计算速度，需386年。





我院吕志鹏教授团队斩获EDA国际算法竞赛冠军（ ICCAD 2021 ）

EDA作为一切芯片设计活动的起点，也被誉为“芯片之母”。吕志鹏教授团队设计的启发式优化算法，在冗余导线检测、布线环路消除、并行化邻域评估加速、布局调整最优移动区域识别等多项关键技术上实现了突破。根据ICCAD 2021会议公布的竞赛结果，该团队设计的算法在所有测试算例上均达到了竞赛中的最优结果。据悉，这是吕志鹏教授团队首次参加该项大赛。本届CAD Contest算法竞赛共有来自12个国家/地区的137支队伍参与，包括众多国内外知名高校与研究机构，如加州伯克利分校、东京大学、台湾国立清华大学、台湾大学、香港中文大学、复旦大学等。

三、算法策略选择【示例】TSP问题

问题的难解性

◆TSP**问题的难解性**：随着城市数量**n**的上升，TSP问题的遍历计算量**(n-1)！** 剧增，计算资源将难以承受。

◆2001年解决了德国**15112**个城市的TSP问题，使用了美国Rice大学和普林斯顿大学之间互连的、速度为500MHz的Compaq EV6 Alpha 处理器组成的**110台计算机**，所有计算机花费的时间之和为**22.6年**。

An optimal TSP tour through Germany's 15 largest cities. It is the shortest among 43 589 145 600 possible tours visiting each city exactly once.

三、算法策略选择【示例】TSP问题

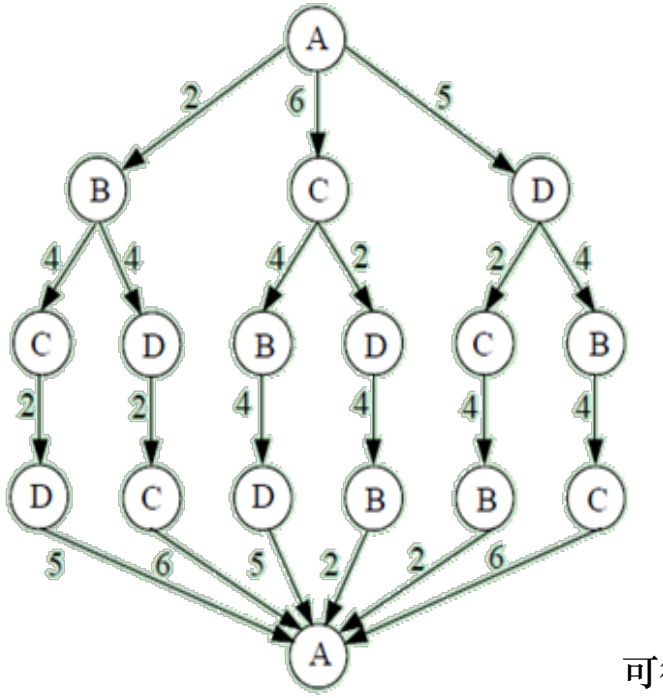
寻找其他策略降低计算量

TSP问题，有没有其他求解算法呢？

◆**最优解** vs. **可行解**

◆不同的算法设计策略：

- ✓ 遍历、搜索算法
- ✓ 分治算法
- ✓ 贪心算法
- ✓ 动态规划算法
- ✓ ……



可行解

最优解

路径：ABCDA 总距离：13　路径：ABDCA 总距离：14

路径：ACBDA 总距离：19　路径：ACDBA 总距离：14

路径：ADCBA 总距离：13　路径：ADBCA 总距离：19

TSP问题的可行解与最优解示意

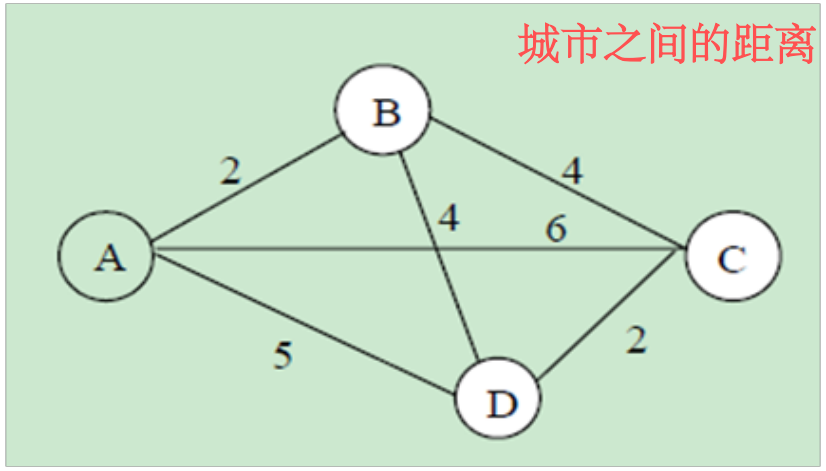
三、算法策略选择【示例】TSP问题

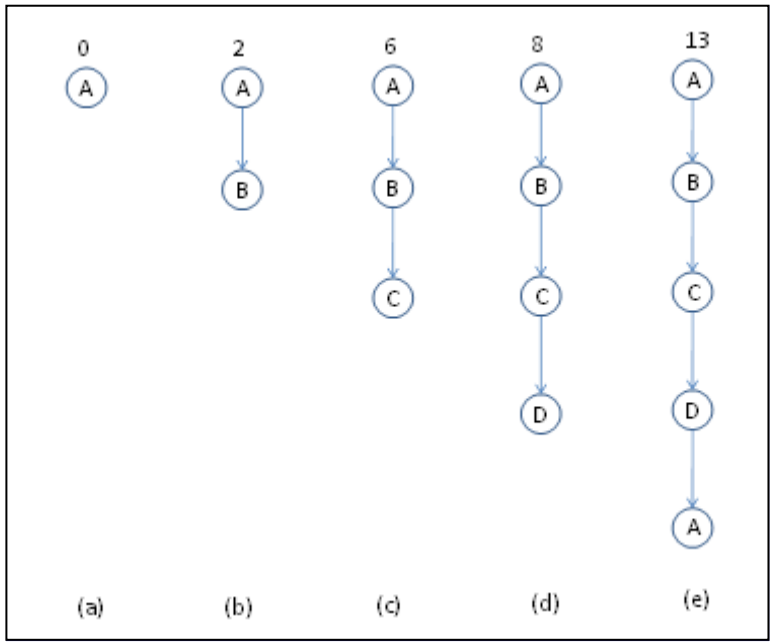
另一种策略：贪心算法

贪心算法是一种算法策略。基本思想“今朝有酒今朝醉”:**一定要做当前情况下的最好选择，否则将来可能会后悔，故名“贪心”。**

✓从某一个城市开始，每次选择一个城市，直到所有城市都被走完。

✓**每次在选择下一个城市的时候，只考虑当前情况，保证迄今为止经过的路径总距离最短。**





第9讲 算法：程序与计算系统之灵魂

一、算法的概念与特征

二、问题与数学建模

三、算法策略选择（算法思想）

四、算法的数据结构及其操作

五、算法的控制结构及其表达方法

六、算法的程序设计

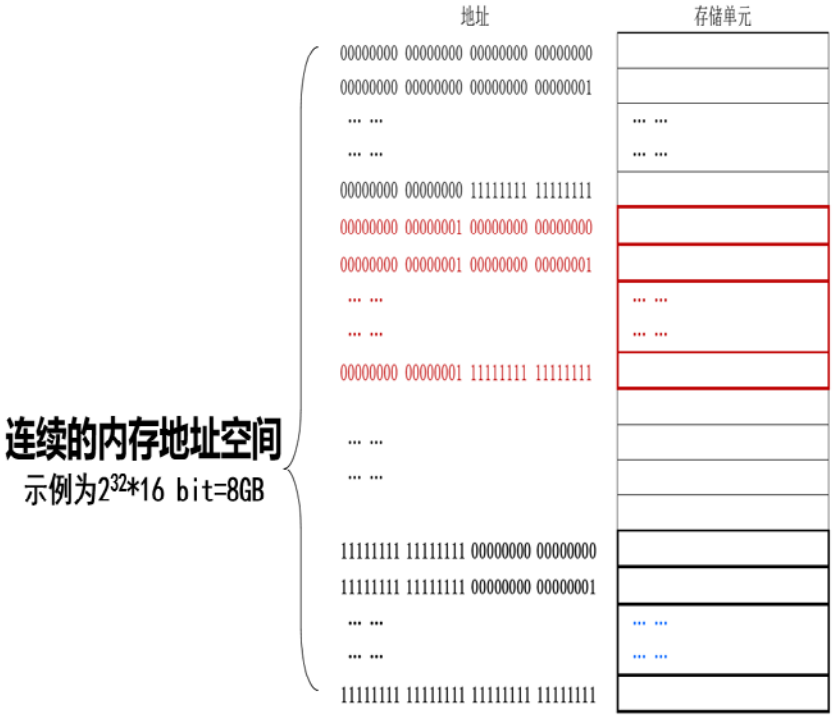
七、算法分析

TSP问题
求解为例

算法的数据结构及其操作

为什么需要数据结构

数据结构是数据的逻辑结构、存储结构及其操作的总称，它提供了问题求解/算法的数据操纵机制。



算法的数据结构及其操作

变量 与 存储单元

	变量名	变量值
Int X=23;	*x*	00000000 00010111
Char Y= 'AB;'	*y*	01000001 01000010
Char Z= 'ABCD';	*Z*	01000001 01000010
		01000011 01000100
		00000000

存储地址	存储内容
00000000 00000001	00000000 00010111
00000000 00000010	01000001 01000010
00000000 00000011	01000001 01000010
00000000 00000100	01000011 01000100
00000000 00000101	00000000
00000000 00000110	

算法的数据结构及其操作

变量、指针变量 与 存储单元

String *P="ABCD";	P
String v = "ABCD";	v *p

存储地址	存储内容	
00000000 00000000	00000100	00001000
00000000 00000001	00000000	00000100
00000000 00000010	00001100	00001010
00000000 00000011	00000000	00000000
00000000 00000100	01000001	01000010
00000000 00000101	01000011	01000100
00000000 00000110	00000000	00000000
00000000 00000111	01000001	01000010
00000000 00001000	01000011	01000100
00000000 00001001	00000000	00000000

算法的数据结构及其操作

不同的变量类型 与 存储单元

用名字表示的存储地址，即变量名	存储地址	存储内容(即变量值)
Mark	00000000 00000000 00000000 00000001 00000000 00000010 00000000 00000011	(注：可通过赋值发生改变)
Sum	00000000 00000100 00000000 00000101	(注：可通过赋值发生改变)
Distance	00000000 00000110 00000000 00000111	(注：可通过赋值发生改变)

算法的数据结构及其操作

多元素变量 与 存储单元

◆**向量**或**列表**是有序数据的集合型变量，向量中的每一个元素都属于同一个数据类型，用一个统一的向量名和下标来唯一的确定向量中的元素。在程序设计语言中，又称为**数组**。

◆向量名通常表示该向量的起始存储地址，而向量下标表示所指向元素相对于起始存储地址的偏移位置。

向量实例

82	Mark[0]
95	Mark[1]
100	Mark[2]
60	Mark[3]
80	Mark[4]

编写求上述数组中值的平均值的程序

```
n = 4;
Sum=0;
For  J =0 to n Step 1
{    Sum = Sum + mark[ J ];
        }
Next  J
Avg = Sum/(n+1);
```

用变量名和元素位置共同表示存储地址，即向量		存储地址	存储内容(即变量值)
Mark	[0]	00000000 00000000 00000000 00000001	(注：82 的 4 字节二进制数可通过赋值发生改变)
	[1]	00000000 00000010 00000000 00000011	(注：95 的 4 字节二进制数可通过赋值发生改变)
	[2]	00000000 00000100 00000000 00000101	(注：100 的 4 字节二进制数可通过赋值发生改变)
	[3]	00000000 00000110 00000000 00000111	(注：60 的 4 字节二进制数可通过赋值发生改变)
	[4]	00000000 00001000 00000000 00001001	(注：80 的 4 字节二进制数可通过赋值发生改变)

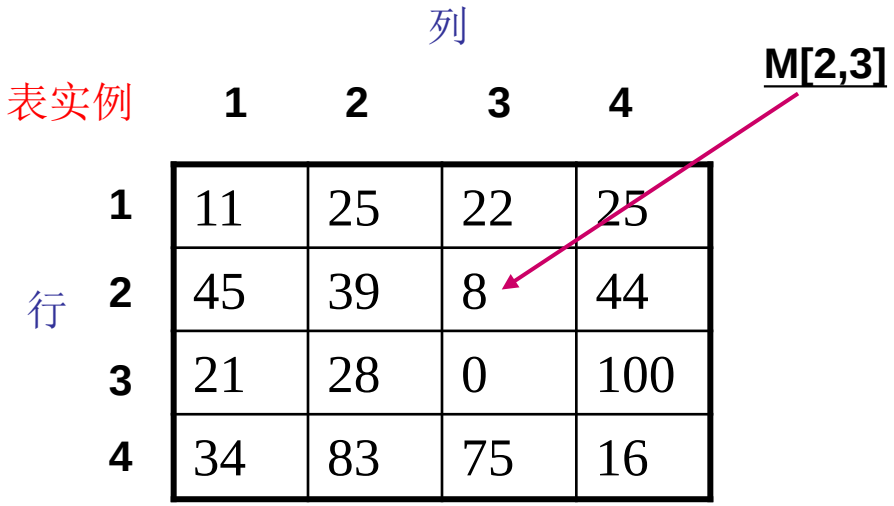
向量存储实例

多元素变量使得程序可通过下标来操作变量中的每一个元素

算法的数据结构及其操作

多元素变量 与 存储单元

◆**矩阵**或**表**是按行按列组织数据的集合型变量，通常是一个二维向量，可表示为如M[2,3]或M[2][3]形式，即用符号名加两个下标来唯一确定表中的一个元素，前一下标为行序号，后一下标为列序号。系统会自动将其转换为对应的存储地址，找到相应的存储单元。在程序设计语言中，矩阵或表是一个多维数组变量。



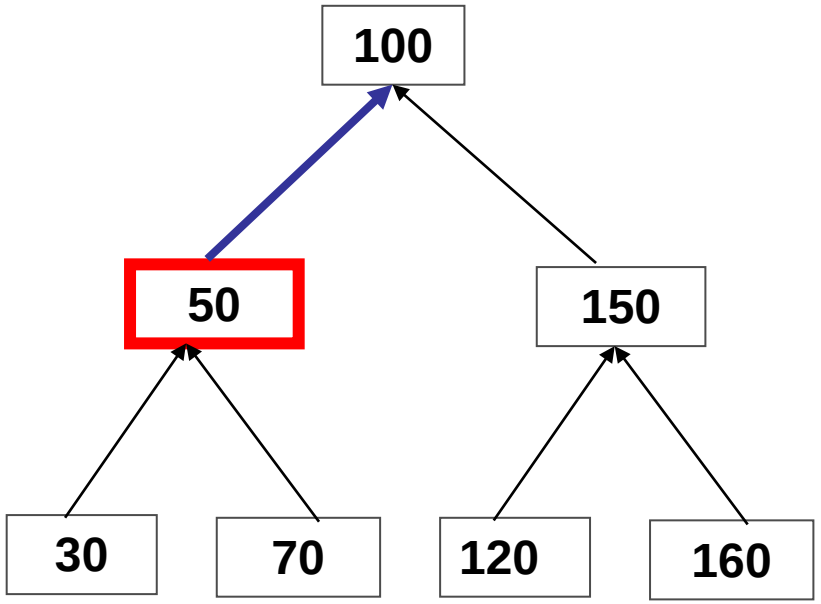

```
Sum=0;
For  I =1 to 4 Step 1
{   For  J =1 to 4 Step 1
    { Sum = Sum + M[I][J];  }
    Next  J
}
Next  I
Avg = Sum/16;
```

逻辑上是二维的按行、列下标来操作一个元素，如M[2,3]或M[2][3]；物理上仍旧是一维存储的，由“表起始地址+ (行下标-1)*列数+(列下标-1)”。这种转换可由系统自动完成，程序中只需按下标操作即可，即如M[2][3]

算法的数据结构及其操作

典型的数据结构-- “树” ：一种存储方法

“树”的**存储结构**: 一个存储单元存储 “数据元素”；一个存储单元存储“指针”，指示数据元素之间的逻辑关系



数据的逻辑结构

数据的存储结构

数据元素

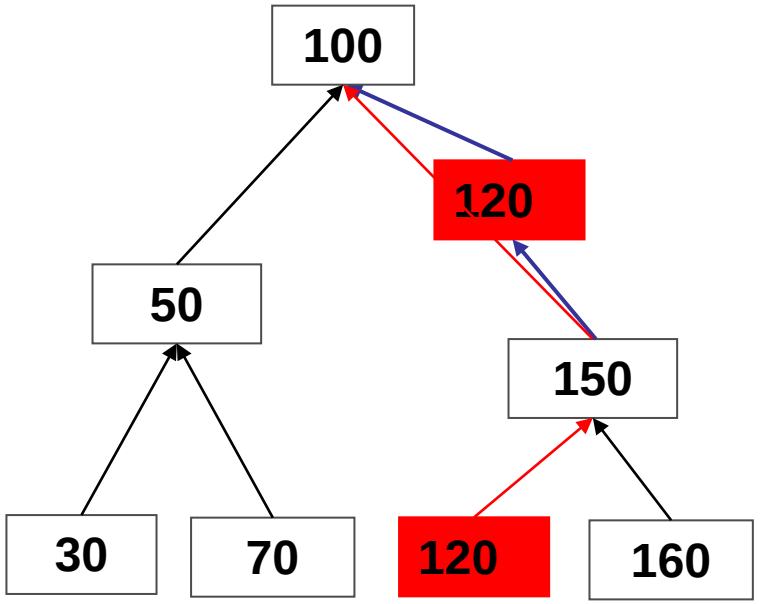
对应数据元素的指针—指向该元素的父元素

存储地址	存储内容(即变量值)	注释
00000000 00000001	00000000 01100100	第 1 个数据元素 100
00000000 00000010	00000000 00000000	第 1 个数据元素的指针
00000000 00000011	00000000 00110010	第 2 个数据元素 50
00000000 00000100	00000000 00000001	第 2 个数据元素的指针
00000000 00000101	00000000 10100110	第 3 个数据元素 150
00000000 00000110	00000000 00000001	第 3 个数据元素的指针
00000000 00000111	00000000 00011110	第 4 个数据元素 30
00000000 00001000	00000000 00000011	第 4 个数据元素的指针
00000000 00001001	00000000 01000110	第 5 个数据元素 70
00000000 00001010	00000000 00000011	第 5 个数据元素的指针
00000000 00001011	00000000 01111000	第 6 个数据元素 120
00000000 00001100	00000000 00000101	第 6 个数据元素的指针
00000000 00001101	00000000 10100000	第 7 个数据元素 160
00000000 00001110	00000000 00000101	第 7 个数据元素的指针
00000000 00001111		

算法的数据结构及其操作

典型的数据结构-- “树”：一种存储方法

存储结构中，通过指针的变化，不改变数据元素的存储，但却改变了数据元素之间的逻辑关系



数据的逻辑结构

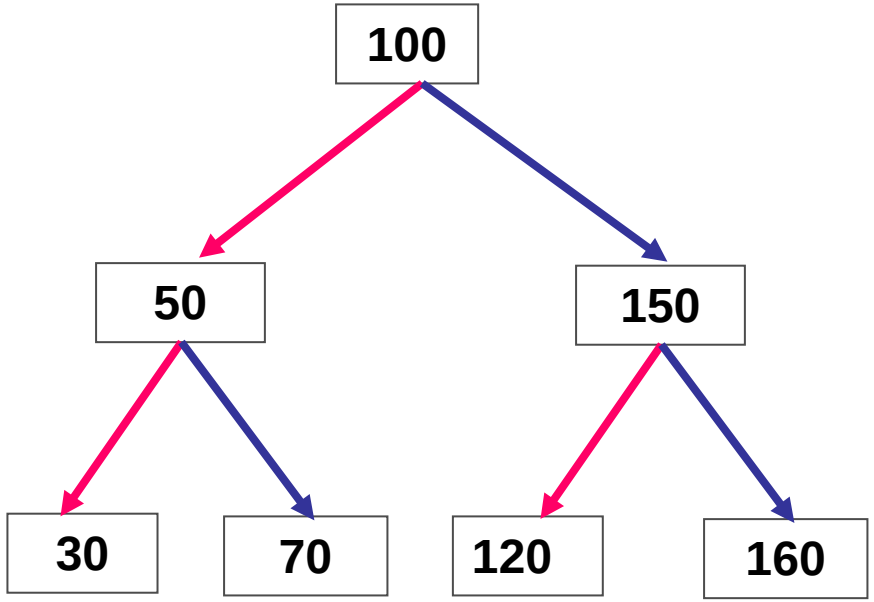
数据的存储结构

存储地址	存储内容(即变量值)	注释
00000000 00000001	00000000 01100100	第 1 个数据元素 100
00000000 00000010	00000000 00000000	第 1 个数据元素的指针
00000000 00000011	00000000 00110010	第 2 个数据元素 50
00000000 00000100	00000000 00000001	第 2 个数据元素的指针
00000000 00000101	00000000 10100110	第 3 个数据元素 150
00000000 00000110	00000000 00001011	第 3 个数据元素的指针
00000000 00000111	00000000 00011110	第 4 个数据元素 30
00000000 00001000	00000000 00000011	第 4 个数据元素的指针
00000000 00001001	00000000 01000110	第 5 个数据元素 70
00000000 00001010	00000000 00000011	第 5 个数据元素的指针
00000000 00001011	00000000 01111000	第 6 个数据元素 120
00000000 00001100	00000000 00000001	第 6 个数据元素的指针
00000000 00001101	00000000 10100000	第 7 个数据元素 160
00000000 00001110	00000000 00000101	第 7 个数据元素的指针
00000000 00001111		

算法的数据结构及其操作

典型的数据结构-- “树”：另一种存储方法

“树”的另一种**存储结构**，用两个指针表达数据之间的逻辑关系，一个指向其左数据元素，一个指向其右数据元素。



数据的逻辑结构

数据的存储结构

数据元素

对应数据元素的左指针—指向该元素的左侧子结点

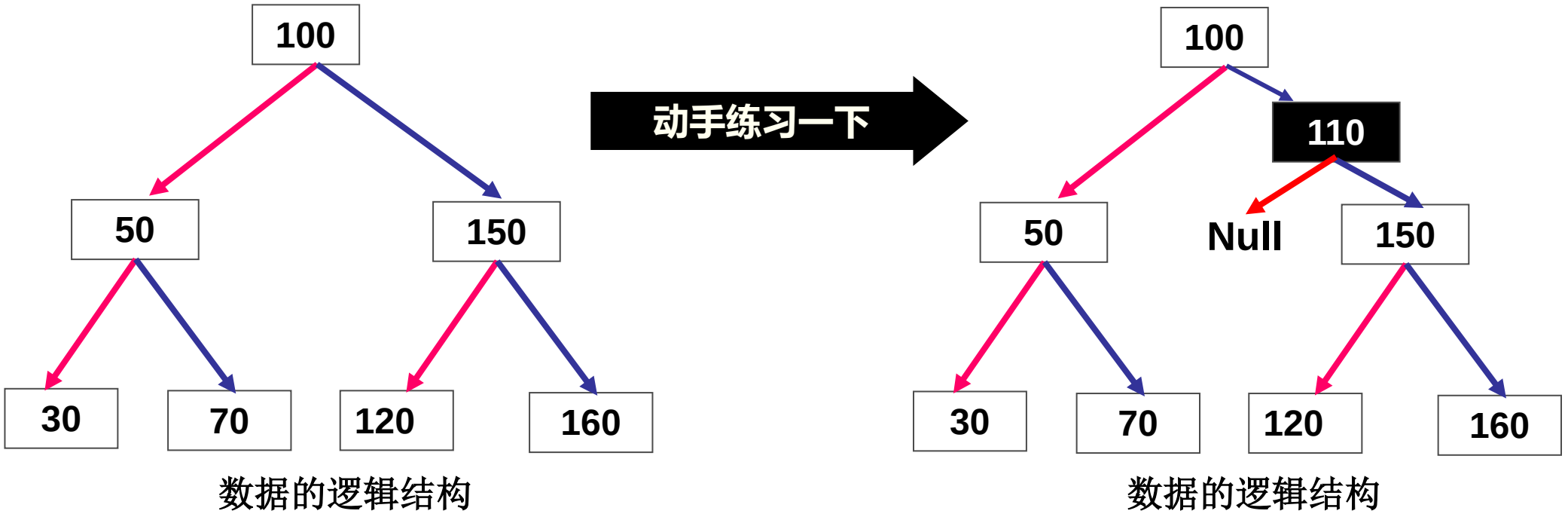
对应数据元素的右指针—指向该元素的右侧子结点

	存储地址	存储内容(即变量值)	注释
TreeElement	00000000 00000001	00000000 01100100	第 1 个数据元素 100
	00000000 00000010	00000000 00110010	第 2 个数据元素 50
	00000000 00000011	00000000 10100110	第 3 个数据元素 150
	00000000 00000100	00000000 00011110	第 4 个数据元素 30
	00000000 00000101	00000000 01000110	第 5 个数据元素 70
	00000000 00000110	00000000 01111000	第 6 个数据元素 120
	00000000 00000111	00000000 10100000	第 7 个数据元素 160
	00000000 00001000		
LeftPointer	00000000 00001001	00000000 00000010	第 1 个数据元素的左指针
	00000000 00001010	00000000 00000100	第 2 个数据元素的左指针
	00000000 00001011	00000000 00000110	第 3 个数据元素的左指针
	00000000 00001100	00000000 00000000	第 4 个数据元素的左指针
	00000000 00001101	00000000 00000000	第 5 个数据元素的左指针
	00000000 00001110	00000000 00000000	第 6 个数据元素的左指针
	00000000 00001111	00000000 00000000	第 7 个数据元素的左指针
RightPointer	00000000 00010000	00000000 00000011	第 1 个数据元素的右指针
	00000000 00010001	00000000 00000101	第 2 个数据元素的右指针
	00000000 00010010	00000000 00000111	第 3 个数据元素的右指针
	00000000 00010011	00000000 00000000	第 4 个数据元素的右指针
	00000000 00010100	00000000 00000000	第 5 个数据元素的右指针
	00000000 00010101	00000000 00000000	第 6 个数据元素的右指针
	00000000 00010110	00000000 00000000	第 7 个数据元素的右指针

数据结构不同，数据之间的操作方法也是不同的

算法的数据结构及其操作

练习一下...



数据的逻辑结构　　数据的逻辑结构

算法的数据结构及其操作【示例】TSP问题

问题求解相关数据如何存储?

城市映射为编号: A---1, B---2, C---3, D---4

城市间距离关系: 表或二维数组D，用D[i][j]或D[i,j]来确定欲处理的每一个元素

城市编号	1	2	3	4
1		2	6	5
2	2		4	4
3	6	4		2
4	5	4	2	

D

D[2][3]

访问路径/解: 一维数组S，用S[j]来确定每一个元素

S	1	4	3	2
	S[1]	S[2]	S[3]	S[4]

{A->D->C->B->A}

第9讲 算法：程序与计算系统之灵魂

一、算法的概念与特征

二、问题与数学建模

三、算法策略选择（算法思想）

四、算法的数据结构及其操作

五、算法的控制结构及其表达方法

六、算法的程序设计

七、算法分析

TSP问题
求解为例

算法的控制结构及其表达方法

算法的类自然语言表达方法

算法与程序的基本控制结构

✓**顺序结构**：“**执行A，然后执行B**”，是按顺序执行一条条规则的一种结构 。

✓**分支结构**：“**如果Q成立，那么执行A，否则执行B**”，Q是某些逻辑条件，即按条件判断结果决定执行哪些规则的一种结构 。

✓**循环结构**：控制指令或规则的多次执行的一种结构---迭代(iteration)

◆**循环结构又分为有界循环结构和条件循环结构。**

✓**有界循环**：“**执行A指令N次**”，其中N是一个整数。

✓**条件循环**：某些时候称为无界循环，“**重复执行A直到条件Q成立**”或“**当Q成立时反复执行A**”，其中Q是条件。

算法的控制结构及其表达方法

示例

【示例】sum=1+2+3+4+…+n求和问题的算法描述

Start of the algorithm(算法开始)

(1)输入N的值；

(2)设 i 的值为1；sum的值为0；

(3)如果 i<=N，则执行第(4)步，否则转到第(7)步执行；

(4)计算 sum + i，并将结果赋给sum；

(5)计算 i+1，并将结果赋给i;

(6)返回到第3步继续执行；

(7)输出sum的结果。

End of the algorithm(算法结束)

自然语言表示的算法容易出现二义性、不确定性等问题

算法的控制结构及其表达方法

算法的程序流程图表达方法

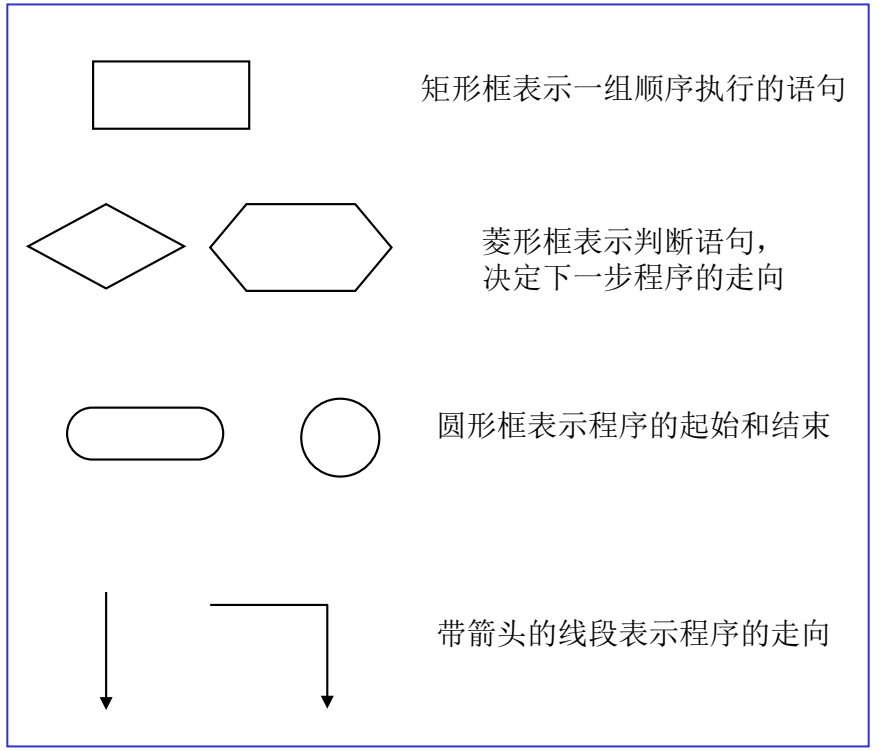
流程图的基本表示符号

✓**矩形框**：表示一组顺序执行的规则或者程序语句。

✓**菱形框**：表示条件判断，并根据判断结果执行不同的分支。

✓**圆形框**：表示算法或程序的开始或结束。

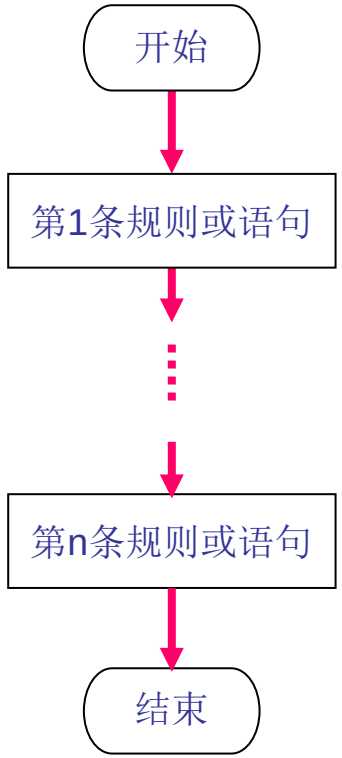
✓**箭头线**：表示算法或程序的走向。



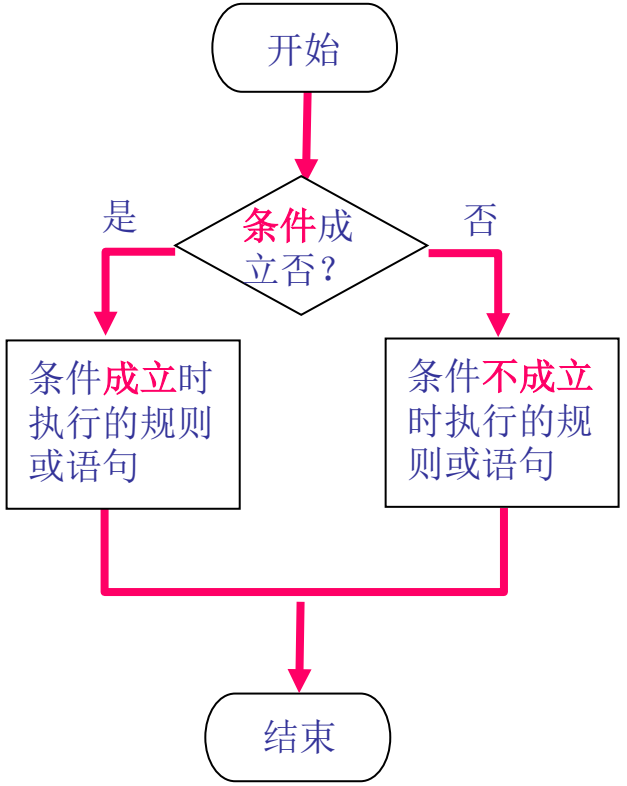
算法的控制结构及其表达方法

算法的程序流程图表达方法

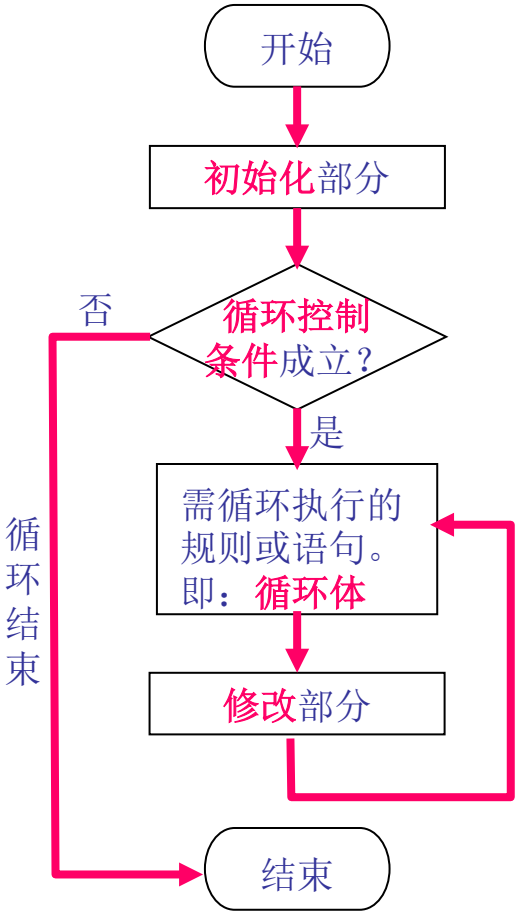
三种控制结构的流程图表达



顺序结构的流程图



分支结构的流程图



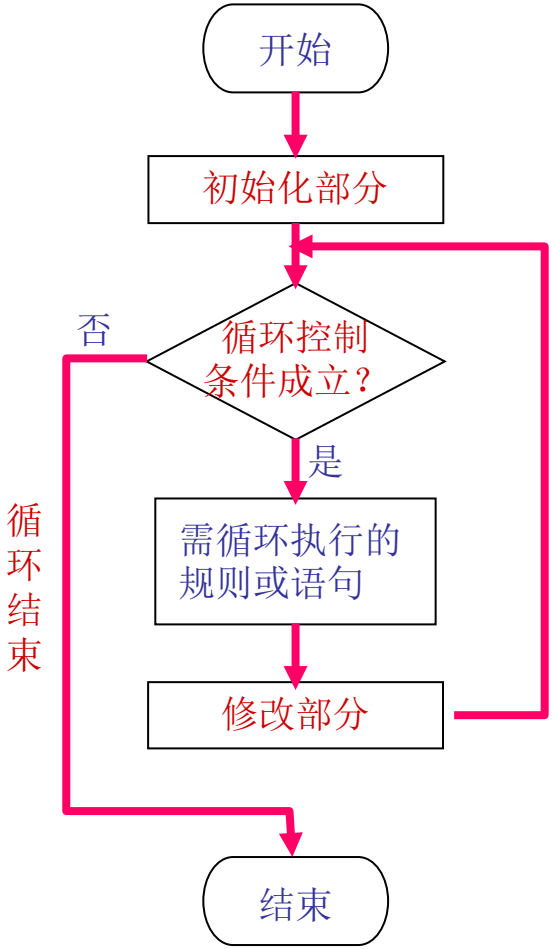
循环结构的流程图

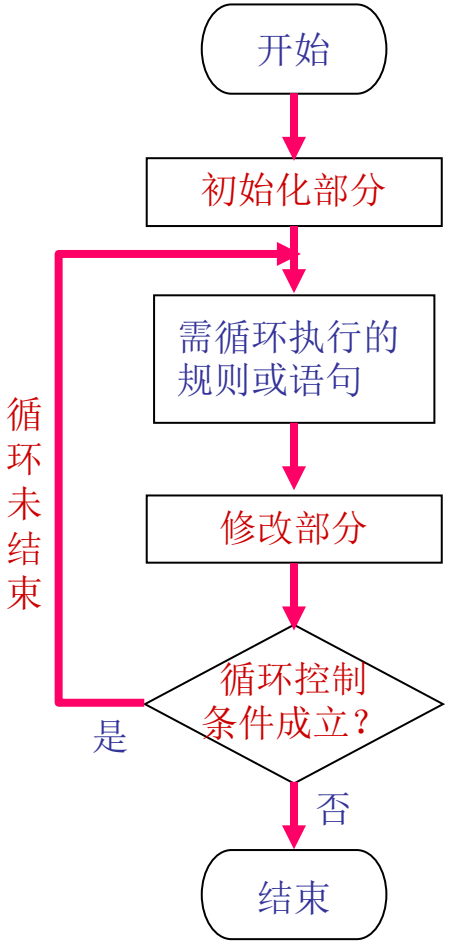
算法的控制结构及其表达方法

算法的程序流程图表达方法

循环结构的两种情况的流程图表达

有界循环结构的流程图(循环次数确定)



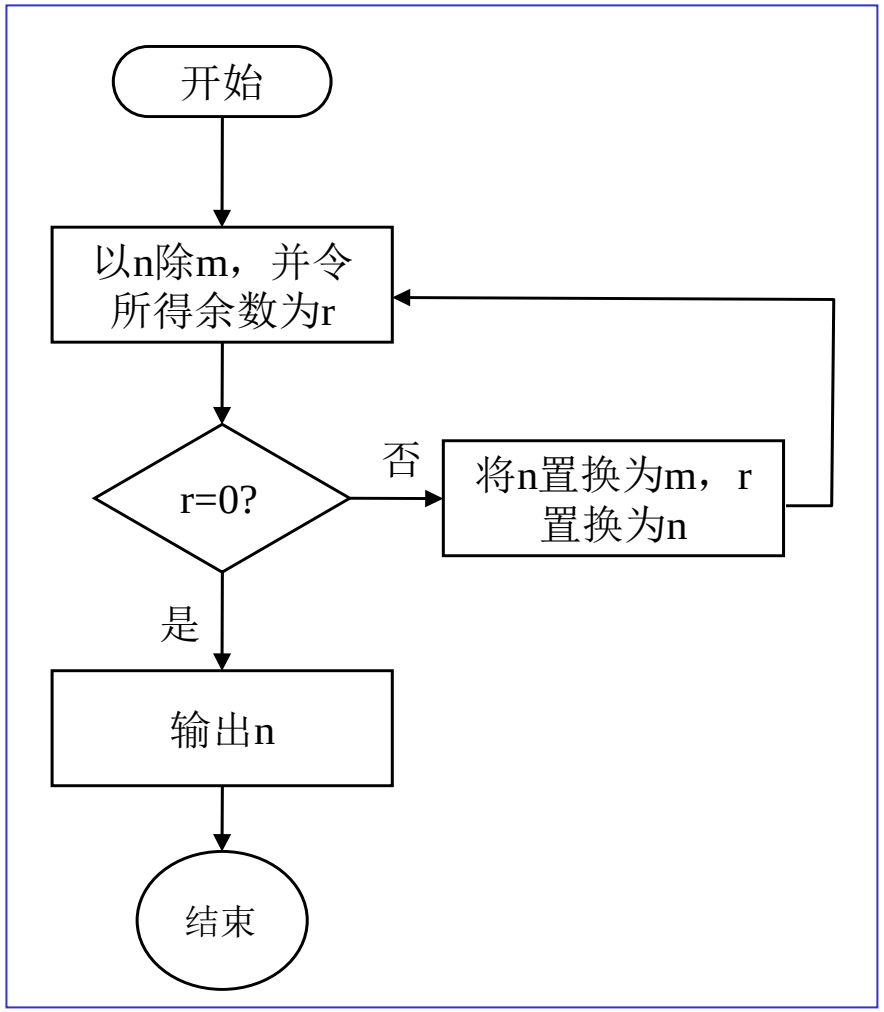


条件循环结构的流程图(循环次数不确定)

算法的控制结构及其表达方法

示例

【示例】欧几里德算法流程图



算法的控制结构及其表达方法【示例】TSP问题

求解TSP问题的遍历算法的表达

	S[1]	S[2]	S[3]	S[4]
S	1	2	3	4
	1	2	4	3
	1	3	2	4
	1	3	4	2
	.....			

Distance =
D[S[4]][S[1]] +
$\sum$ D[S[i]][S[i+1]]
i=1 to 3

当前最短路径 MS

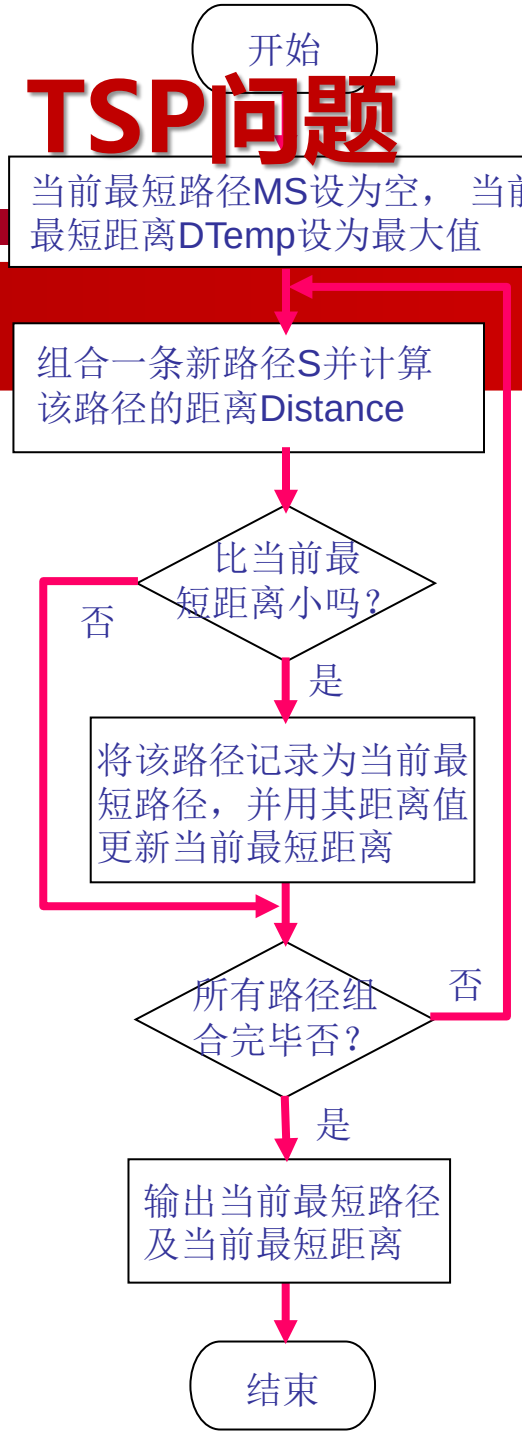
1	2	3	4

当前最短距离 DTemp

D

城市编号	1	2	3	4
1		2	6	5
2	2		4	4
3	6	4		2
4	5	4	2	

将思想/策略转变为“步骤”



算法的控制结构及其表达方法【示例】TSP问题

求解TSP问题的贪心算法的表达

✓依次访问过的城市编号被记录在S[1], S[2], …, S[N}中,即第I 次访问的城市记录在S[I]中。

✓Step(1)：从1号城市开始访问，将城市编号1赋值给S[1]。

✓Step(6)：第I次访问的城市为城市j，其距第I-1次访问的城市的距离最短。

S

S[1]	S[2]	S[3]	S[4]
1			

I = 2, 3, 4 ---当前要找第几个

j = 1, 2, 3, 4 ---城市号

D

城市编号	1	2	3	4
1		2	6	5
2	2		4	4
3	6	4		2
4	5	4	2	

Start of the Algorithm

(1) S[1]=1;

(2) Sum=0;

(3) 初始化距离数组D[N, N];

(4) I=2;

(5) 从所有未访问过的城市中查找距离S[I-1]最近的城市j;

(6) S[I]=j;

(7) I=I+1;

(8) Sum=Sum+Dtemp;

(9) 如果I<=N，转步骤(5)，否则，转步骤(10);

(10)Sum=Sum+D[1, j];

(11)逐个输出S[N]中的全部元素;

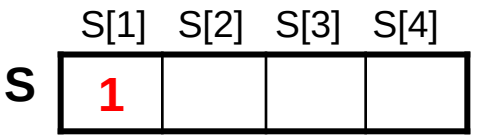
(12)输出Sum。

End of the Algorithm

算法的控制结构及其表达方法【示例】TSP问题

求解TSP问题的贪心算法的表达

前述第5步“从所有未访问过的城市中查找距离S[I-1]最近的城市j”还是不够明确，需要进一步细化



I = 2, 3, 4 ---当前要找第几个

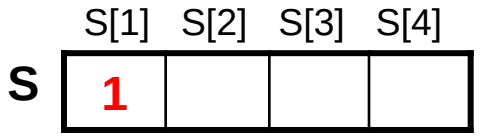
L = 1, 2, ⋯,I-1 ---从第1个访问过的，到第I-1个访问过的

K = 2, 3, 4 ---城市号

(5.1)K=2;
(5.2)将Dtemp设为一个大数(比所有两个城市之间的距离都大)
  (5.3)L=1;
  (5.4)如果S[L]==K，转步骤5.8；//该城市已出现过，跳过
  (5.5)L=L+1;
  (5.6)如果L<I，转5.4;
(5.7)如果D[K,S[I-1]]<Dtemp,则j=K; Dtemp＝D[K,S[I-1]];
(5.8)K=K+1;
(5.9)如果K<=N，转步骤5.3。

算法的控制结构及其表达方法【示例】TSP问题

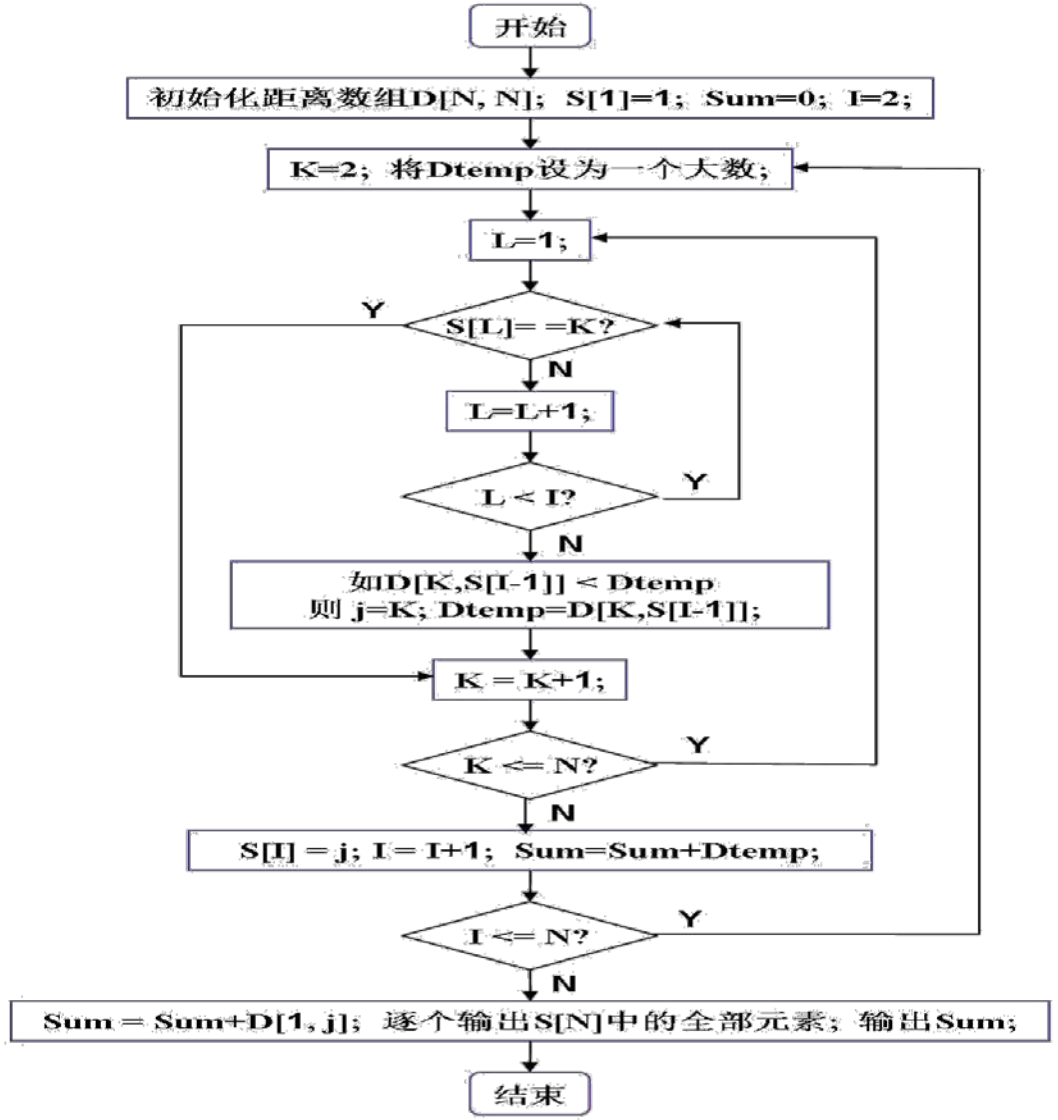
求解TSP问题的贪心算法的表达



I = 2, 3, 4 ---当前要找第几个

K = 2, 3, 4 ---城市号

L = 1, 2, …, I-1 ---从第1个访问过的，到第I-1个访问过的



算法的控制结构及其表达方法【示例】TSP问题

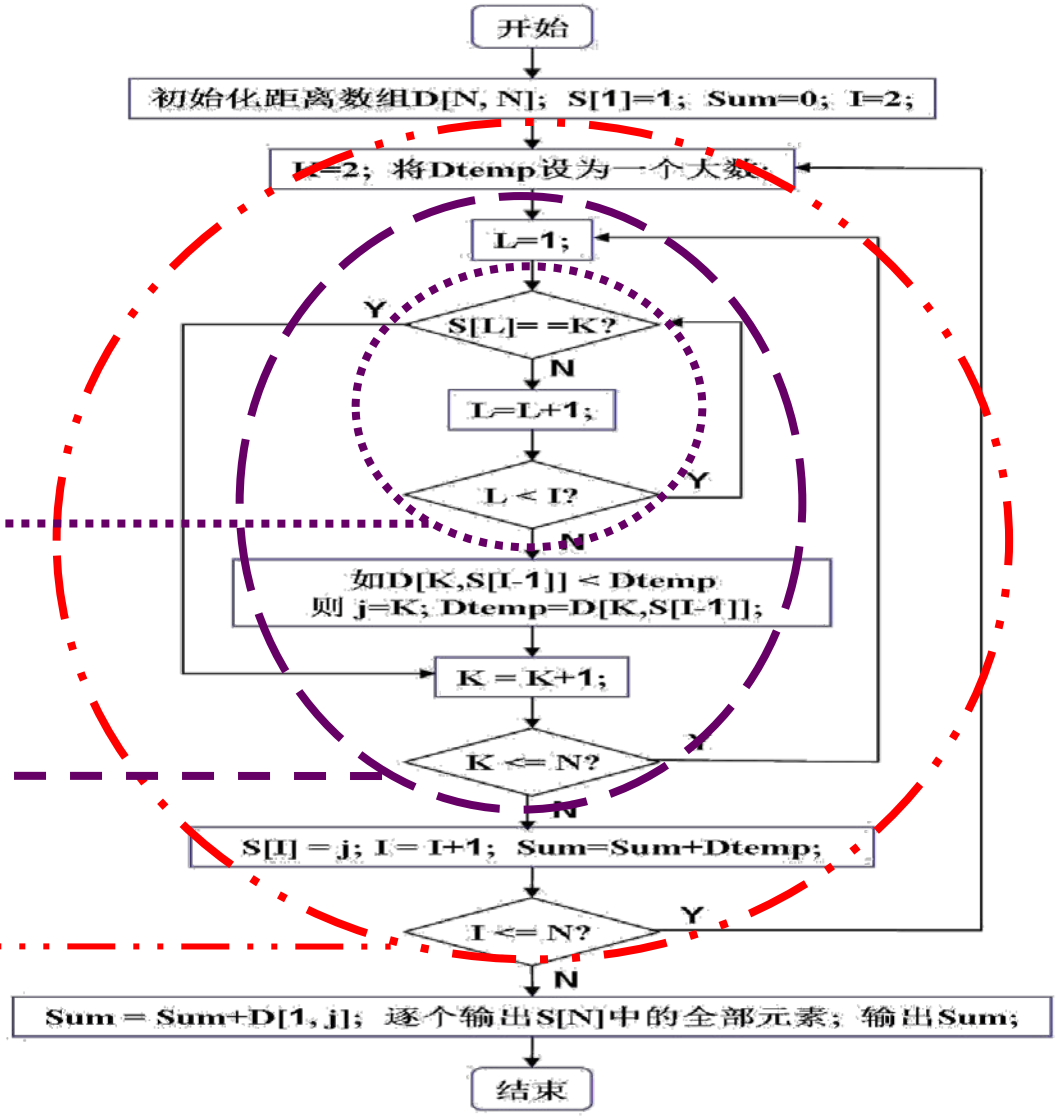
阅读并理解算法的能力

◆计算学科的学生不仅能够设计算法，而且会描述和表达算法，更要能读懂算法。

内层循环，L从1至I-1，循环判断第K号城市是否是已访问过的城市，如是则不参加最小距离的比较；

中层循环，K从2号城市至N号城市循环，判断D[K, S[I-1]]是否是最小值，j记录了最小距离的城市号K。

外层循环，I从2至N循环；I-1个城市已访问过，正在找与第I-1个城市最近距离的城市；已访问过的城市号存储在S[]中。



第9讲 算法：程序与计算系统之灵魂

一、算法的概念与特征

二、问题与数学建模

三、算法策略选择（算法思想）

四、算法的数据结构及其操作

五、算法的控制结构及其表达方法

六、算法的程序设计

七、算法分析

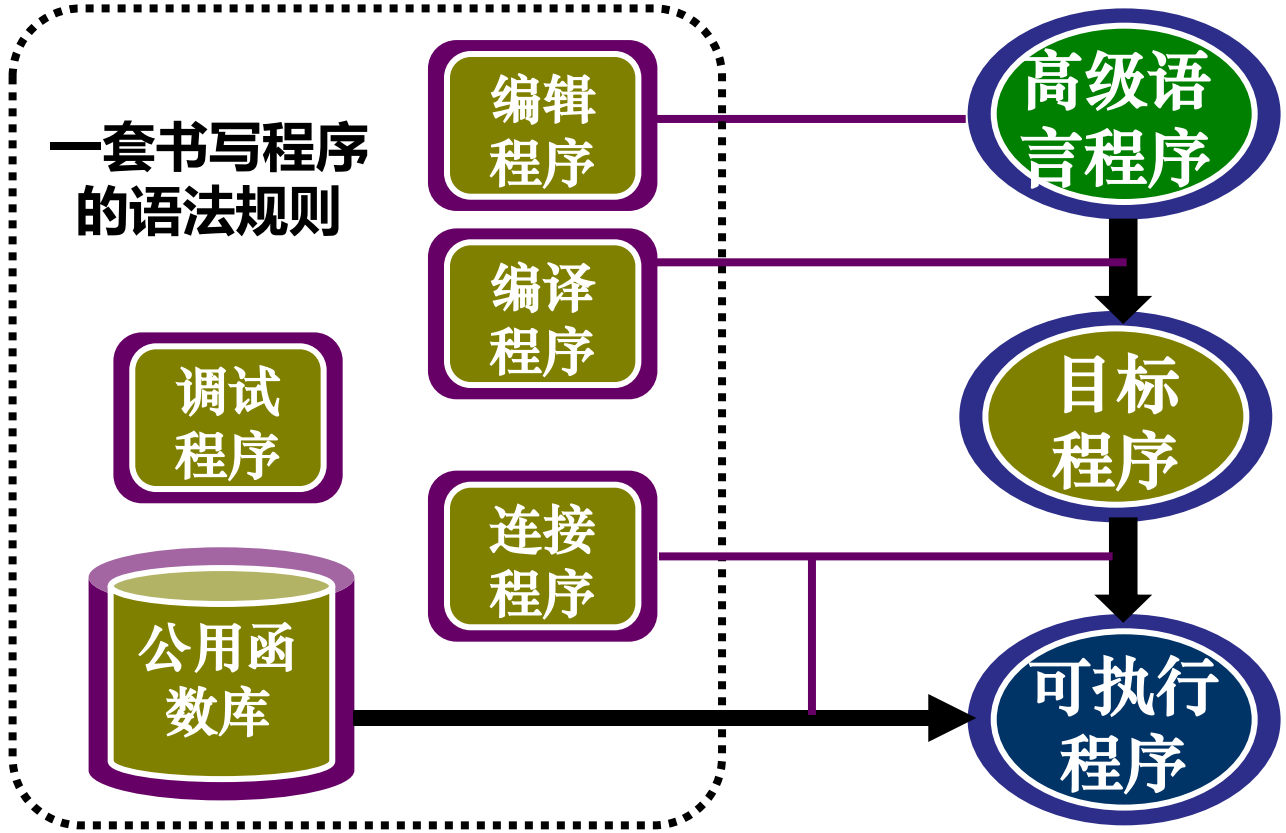
TSP问题
求解为例

算法的程序设计

算法的实现环境与过程

程序设计过程：编辑源程序➔编译➔链接➔执行。

计算机语言程序设计环境：编辑、编译、连接、调试、运行一体化平台



算法的程序设计【示例】TSP问题

实现算法的程序：阅读与理解

```
main() {     判断城市K是否是已访问过的城市
…
//前面已为 K 和 I 赋过值
L=0;    Found=0;
do {
          if(S[L]==K)
          { Found=1; break; }
          else L=L+1;
     } while(L<I);          //L从1到I循环
if Found==0 { //城市K未出现过 }
else { //城市K出现过 }
…
}
```

- K从1,…,n-1是城市的编号
- I是至今已访问过的城市数
- S[0]至S[I-1]中存储的是已访问过的城市的编号

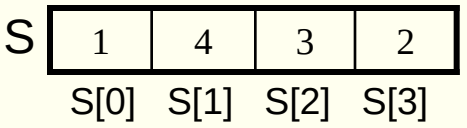
S	1	4	3	2
	S[0]	S[1]	S[2]	S[3]

算法的程序设计【示例】TSP问题

实现算法的程序：阅读与理解

```
main() {  …     寻找下一个未访问过的城市j,且其距当前访问城市S[I-1]距离最短
    K=1;   Dtemp=10000;
    do{  // K从1到n-1循环
        L=0;   Found=0;
        do{
            if(S[L]==K)
            {   Found=1;   break;   }
            else L=L+1;
         } while(L<I);     //L从1到I循环
        if (Found==0 && D[K][S[I-1]]<Dtemp)
        {   j=K;   Dtemp=D[K][S[I-1]];   }
        K=K+1;
    } while(K<n);                        //K从1到n-1循环
    S[I]=j; Sum=Sum+Dtemp;
    … }
```

- K从1,…,n-1是城市的编号
- I是至今已访问过的城市数
- S[I-1]中存储的是当前访问过的城市编号，要找第I个程序



- D[K][S[I-1]]为城市K距当前访问过的城市的距离

算法的程序设计【示例】TSP问题

实现算法的程序：阅读与理解

TSP问题的贪心算法程序实例

```
#include <stdio.h>
#define   n   4
main(){
  int    D[n][n], S[n], Sum, I, j, K, L, Dtemp, Found;
  S[0]=0;  Sum=0;    D[0][1]=2;  D[0][2]=6;  D[0][3]=5;  D[1][0]=2;  D[1][2]=4;
  D[1][3]=4;  D[2][0]=6;  D[2][1]=4;  D[2][3]=2;  D[3][0]=5;  D[3][1]=4;  D[3][2]=2;
  I=1;                    //注意程序是从 0 开始，流程图是从 1 开始的，下同.
  do {                    //I 从 1 到 n-1 循环
      K=1;  Dtemp=10000;
      do{                         //K 从 1 到 n-1 循环
          L=0;  Found=0;
          do{                           //L 从 1 到 I 循环
              if(S[L]==K)
              {  Found=1;  break;  }
              else L=L+1;
            } while(L<I);          //L 从 1 到 I 循环
          if(Found==0 && D[K][S[I-1]]<Dtemp)
          {  j=K;  Dtemp=D[K][S[I-1]];  }
          K=K+1;
      } while(K<n);              //K 从 1 到 n-1 循环
      S[I]=j;  I=I+1;  Sum=Sum+Dtemp;
    } while(I<n);          //I 从 1 到 n-1 循环
  Sum=Sum+D[1][j];
  for(j=0;j<n;j++){  printf("%d,",S[j]);  }  //输出城市编号序列
  printf("\n");                              //换行
  printf("Total Length:%d",Sum);             //输出总距离
}
```

算法的程序设计

实现算法的程序

TSP问题的贪心Python实例

```
def greedy():
    """C语言转python版本"""
    matrix = [  [0.0,   19.02,  34.01,  44.10,  2684,   50.69],
                [19.02, 0.0,    2684,   45.88,  42.57,  2684],
                [34.01, 2684,   0.0,    2684,   54.33,  39.35],
                [44.10, 45.88,  2684,   0.0,    44.40,  11.18],
                [2684,  42.57,  54.33,  44.40,  0.0,    54.56],
                [50.69, 2684,   39.35,  11.18,  54.56,  0.0]]

    S = [0] * len(matrix)    # 有n个0的数组
    n = 6
    I = 1
    j = 0
    K = 0
    L = 0
    Dtemp = 0
    Found = 0
    Sum = 0
    while I < n:
        # 找第I次的最短访问节点
        K = 1
        Dtemp = 10000
        while K < n:
            # 假设第I次访问的是K，看K是不是已经访问过。。
            L = 0
            Found = 0
            while L < I:
                if S[L] == K:
                    Found = 1
                    break
                L += 1
            if Found == 0 and matrix[K][S[I-1]] < Dtemp:
                j = K
                Dtemp = matrix[K][S[I-1]]
            K += 1
        S[I] = j
        I += 1
        Sum += Dtemp
    Sum += matrix[0][j]
    S.append(0)
    for j in range(len(S)): print("%d\t" % S[j], end='')
    print('\n总距离%.2f' % Sum)
```

第9讲 算法：程序与计算系统之灵魂

一、算法的概念与特征

二、问题与数学建模

三、算法策略选择（算法思想）

四、算法的数据结构及其操作

五、算法的控制结构及其表达方法

六、算法的程序设计

七、算法分析

TSP问题
求解为例

算法分析

算法的模拟与分析

算法是正确的吗?

◆算法的正确性问题

- ✓ 问题求解的过程、方法——算法是正确的吗？算法的输出是问题的解吗？满足问题求解约束的解就是问题的解，但不一定是最优解。
- ✓ 20世纪60年代，美国一架发往金星的航天飞机由于控制程序出错而永久丢失在太空中。

◆ 算法的效果评价

- ✓ 算法的输出是最优解还是可行解？**如果是可行解，与最优解的偏差多大**？

■ 两种评价方法：

- ✓ 证明方法：利用数学方法证明；
- ✓ 仿真分析方法：产生或选取大量的、具有代表性的问题实例，利用该算法对这些问题实例进行求解，并对算法产生的结果进行统计分析。

算法获得的解是最优的吗?

算法分析【示例】TSP问题

TSP问题贪心算法的正确性评价

直观上只需检查算法的输出结果中，每个城市出现且仅出现一次，该结果即是TSP问题的可行解，说明算法正确地求解了这些问题实例

	S[1]	S[2]	S[3]	S[4]
S	1	2	3	4

```
#include <stdio.h>
#define  n  4
main(){
  int   D[n][n], S[n], Sum, I, j, K, L, Dtemp, Found;
  S[0]=0;  Sum=0;   D[0][1]=2;  D[0][2]=6;  D[0][3]=5;  D[1][0]=2;  D[1][2]=4;
  D[1][3]=4;  D[2][0]=6;  D[2][1]=4;  D[2][3]=2;  D[3][0]=5;  D[3][1]=4;  D[3][2]=2;
  I=1;                    //注意程序是从 0 开始，流程图是从 1 开始的，下同.
  do {                    //I 从 1 到 n-1 循环
     K=1;  Dtemp=10000;
     do{                  //K 从 1 到 n-1 循环
        L=0;  Found=0;
        do{               //L 从 0 到 I-1 循环
           if(S[L]==K)
           {  Found=1;  break;  }
           else L=L+1;
        } while(L<I);     //L 从 0 到 I-1 循环
        if(Found==0 && D[K][S[I-1]]<Dtemp)
        {  j=K;  Dtemp=D[K][S[I-1]];  }
        K=K+1;
     } while(K<n);        //K 从 1 到 n-1 循环
     S[I]=j;  I=I+1;  Sum=Sum+Dtemp;
  } while(I<n);           //I 从 1 到 n-1 循环
  Sum=Sum+D[j][0];
  for(j=0;j<n;j++){  printf("%d,",S[j]);  }  //输出城市编号序列
  printf("\n");                              //换行
  printf("Total Length:%d",Sum);             //输出总距离
}
```

算法分析【示例】TSP问题

TSP问题贪心算法的效果评价

✓如果实例的最优解已知（问题规模小或问题已被成功求解），利用统计方法对若干问题实例的算法结果与最优解进行对比分析，即可对其进行效果评价；

✓对于较大规模的问题实例，其最优解往往是未知的，因此，算法的效果评价只能借助于与前人算法结果的比较。

	S[1]	S[2]	S[3]	S[4]
S	1	2	3	4

```
#include <stdio.h>
#define  n  4
main(){
  int   D[n][n], S[n], Sum, I, j, K, L, Dtemp, Found;
  S[0]=0;  Sum=0;   D[0][1]=2;  D[0][2]=6;  D[0][3]=5;  D[1][0]=2;  D[1][2]=4;
  D[1][3]=4;  D[2][0]=6;  D[2][1]=4;  D[2][3]=2;  D[3][0]=5;  D[3][1]=4;  D[3][2]=2;
  I=1;                    //注意程序是从 0 开始，流程图是从 1 开始的，下同.
  do {                    //I 从 1 到 n-1 循环
     K=1;  Dtemp=10000;
     do{                  //K 从 1 到 n-1 循环
        L=0;  Found=0;
        do{               //L 从 0 到 I-1 循环
           if(S[L]==K)
           {  Found=1;  break;  }
           else L=L+1;
        } while(L<I);     //L 从 0 到 I-1 循环
        if(Found==0 && D[K][S[I-1]]<Dtemp)
        {  j=K;  Dtemp=D[K][S[I-1]];  }
        K=K+1;
     } while(K<n);        //K 从 1 到 n-1 循环
     S[I]=j;  I=I+1;  Sum=Sum+Dtemp;
  } while(I<n);           //I 从 1 到 n-1 循环
  Sum=Sum+D[j][0];
  for(j=0;j<n;j++){  printf("%d,",S[j]);  }  //输出城市编号序列
  printf("\n");                              //换行
  printf("Total Length:%d",Sum);             //输出总距离
}
```

算法分析

算法的时间复杂度与空间复杂度

算法获得结果的时间有多长?

◆**时间复杂性:** 如果一个问题的规模是n，解这一问题的某一算法所需要的时间为T(n)，它是n的某一函数， 则T(n)称为这一算法的“时间复杂性”。

◆ **“大O记法”：**

✓基本参数 n——问题实例的规模

✓把复杂性或运行时间表达为n的函数。

✓ “O”表示量级 (order)，允许使用“=”代替“≈”，如$n^2+n+1 =O(n^2)$ 。

◆**空间复杂性:** 算法在执行过程中所占存储空间的大小。

算法分析

算法复杂性分析示例

```
sum=0;                                  （1次）
for( i=1; i<=n; i++)                    （n次）
{  for( j=1; j<=n; j++)                 （n²次）
        { sum++;  }                      （n²次）
}
```

解：$T(n) = 2n^2+n+1 = O(n^2)$

主要关注点：循环的层数

算法分析【示例】TSP问题

TSP问题贪心算法的复杂性分析

✓一个关于n的三重循环。

✓时间复杂度是$O(n^3)$。

```
#include <stdio.h>
#define  n  4
main(){
  int    D[n][n], S[n], Sum, I, j, K, L, Dtemp, Found;
   S[0]=0;  Sum=0;   D[0][1]=2;  D[0][2]=6;  D[0][3]=5;  D[1][0]=2;  D[1][2]=4;
   D[1][3]=4;  D[2][0]=6;  D[2][1]=4;  D[2][3]=2;  D[3][0]=5;  D[3][1]=4;  D[3][2]=2;
   I=1;                    //注意程序是从 0 开始，流程图是从 1 开始的，下同.
   do {                    //I 从 1 到 n-1 循环---将被执行 n-1 次，简记约 n 次
      K=1;  Dtemp=10000;
      do{                  //K 从 1 到 n-1 循环---将被执行(n-1)*(n-1)次，简记约 n²次
         L=0;  Found=0;
         do{                //L 从 1 到 I 循环---将被执行，简记约 n³次
            if(S[L]==K)
            {  Found=1;  break;  }
            else L++;
         } while(L<I);      //L 从 1 到 I 循环
         if(Found==0 && D[K][S[I-1]]<Dtemp)
         {  j=K;  Dtemp=D[K][S[I-1]];  }
         K++;
      } while(K<n);         //K 从 1 到 n-1 循环
      S[I]=j;  I=I+1;  Sum=Sum+Dtemp;
   } while(I<n);            //I 从 1 到 n-1 循环
   Sum=Sum+D[1][j];
   for(j=0;j<n;j++){  printf("%d,",S[j]);  }  //输出城市编号序列
   printf("\n");                              //换行
   printf("Total Length:%d",Sum);             //输出总距离
}
```

n

n^2

n^3

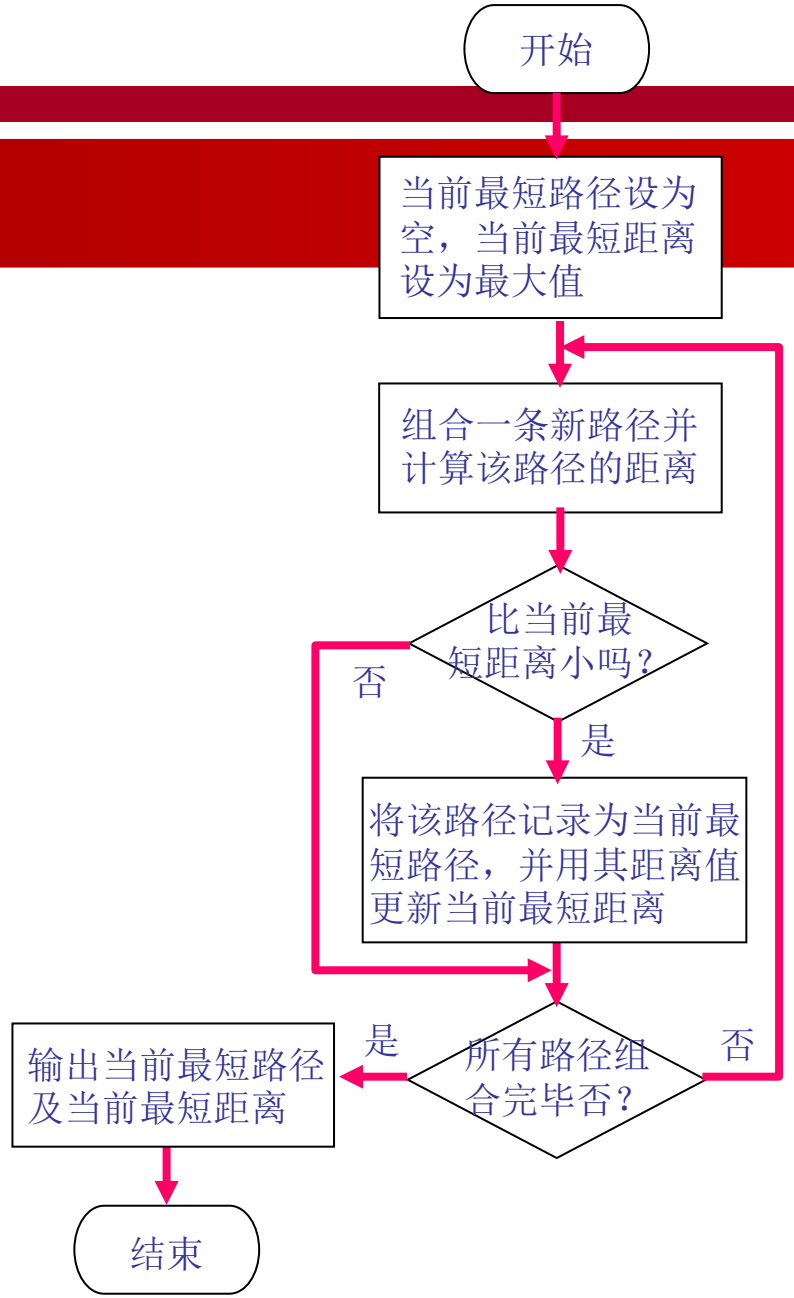
算法分析【示例】TSP问题

TSP问题遍历算法的复杂性分析

✓列出每一条可供选择的路线，计算出每条路线的总里程，最后从中选出一条最短的路线。

✓路径组合数目为(n-1)!

✓时间复杂度是**O((n-1)!)**



算法分析

不同的算法复杂性量级

问题规模n	计算量
10	10！
20	20！
100	100！
1000	1000！
10000	10000！

$20! = 1.216 \times 10^{17}$

$20^3 = 8000$

$O(n^3)$与$O(3^n)$的差别，$O(n!)$与$O(n^3)$的差别

$O(n^3)$	$O(3^n)$
0.2秒	4×10^{28}秒 =1015年

注：每秒百万次，n=60，1015年相当于10亿台计算机计算一百万年

$O(1), O(\log n), O(n), O(n \log n), O(n^b)$ $O(b^n), O(n!)$

算法分析

有限时间内是否能完成算法的执行?

◆当算法的复杂度是一个多项式，如$O(n^2)$时，则对于大规模问题，该算法是可以在有限时间内被计算机完成的。例如，TSP问题的贪心算法 $O(n^3)$ 。

$O(1), O(\log n), O(n), O(n \log n), O(n^b)$

◆当算法的复杂度是用指数函数表示如$O(2^n)$或阶乘函数表示如$O(n!)$，则当n很大（如10000）时，该算法是计算机在有限时间内无法处理的。例如TSP问题的遍历算法O((n-1)!)。

$O(b^n), O(n!)$

算法分析

计算复杂性与算法复杂性

◆**算法复杂性：** 求解问题的某一个算法的复杂性。

◆**计算复杂性：** 能求问题精确解的那个算法的复杂性。

计算复杂性是指问题的一种特性，即利用计算机求解问题的难易性或难易程度

问题的计算复杂性

●计算机在有限时间内能够求解的--**【可求解问题】**

●计算机在有限时间内不能求解的--**【难求解问题】**

●计算机完全不能求解的--**【不可计算问题】**

算法分析【示例】TSP问题

TSP问题的计算复杂性与算法复杂性

- TSP问题的遍历算法的时间复杂度：$O((n-1)!)$
- TSP问题的贪心算法的时间复杂度：$O(n^3)$。

算法复杂性
-某一个算法的复杂性

- TSP问题的计算复杂度：$O((n-1)!)$

计算复杂性
-求精确解的那个算法的复杂性

TSP问题的遍历算法（又称穷举或蛮干）

对解空间中的每一个可能解进行验证，直到所有的解都被验证是否正确，便能得到精确的结果---**精确解**

可能是$O(n!)$或$O(a^n)$

TSP问题的贪心算法

近似解求解算法---**近似解**

应该是$O(n^a)$

Δ = | 近似解 - 精确解 |
满意解： Δ充分小时的近似解

算法分析 --P类问题与NP类问题

证比求易问题

问题的引入——国王求婚 问题

求出48 770 428 433 377 171 （17位数）的一个真因子。

算法1：从2开始逐个除，一秒一个，一天算86400个

答案：223 092 827。

算法2：最小的真因子不会超过9位，全民动员

串行与并行，时间与空间，折衷

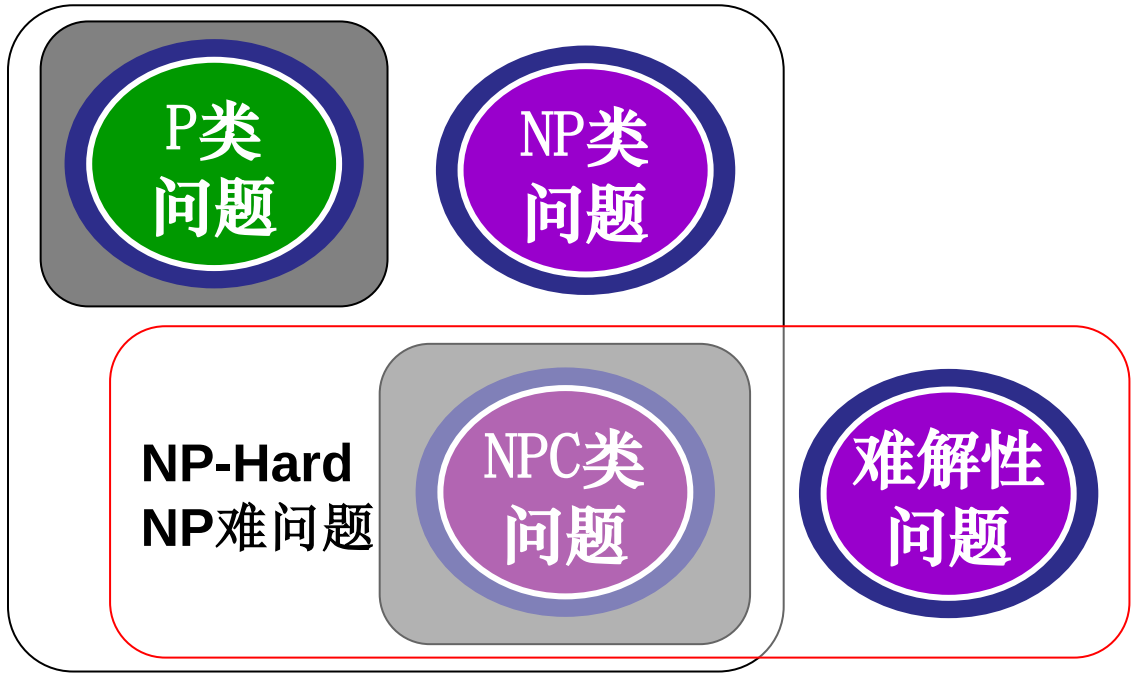
算法分析

可解性问题与难解性问题

■P类问题：多项式问题(Polynomial Problem)，指计算机可以在有限时间内求解的问题，即：可以找出一个呈现$O(n^a)$复杂性算法的问题，其中a为常数。

■NP类问题：非确定性多项式问题(Non-deterministic Polynomial)。有些问题，其答案是无法直接计算得到的，只能通过间接的猜算或试算来得到结果，这就是非确定性问题(Non-deterministic)。虽然在多项式时间内难于求解但不难判断给定一个解的正确性的问题，即：在多项式时间内可以由一个算法验证一个解是否正确的非确定性问题，就是NP类问题。

■NPC问题：如果NP问题的所有可能答案都可以在多项式时间内进行正确与否的验算，且枚举可能答案的时间复杂性在指数级别，就叫做完全非确定性多项式问题(NP-Complete)。

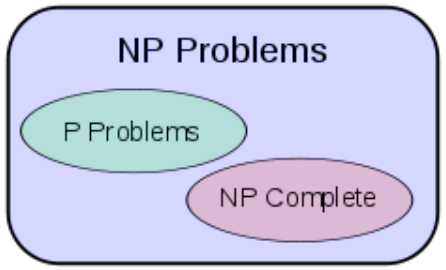


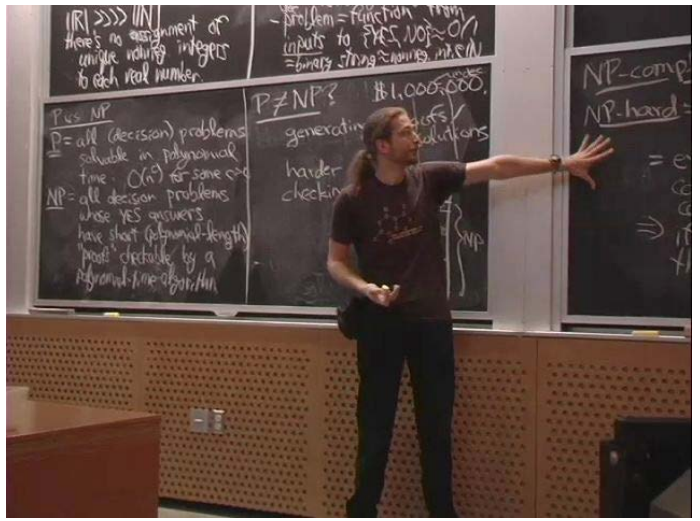
问：加密算法应设计成一个什么问题呢？

算法分析

可解性问题与难解性问题

- 所有可以在多项式时间内求解的问题，称为【**P类问题**】
- 所有在多项式时间内可以验证的问题，称为【**NP类问题**】，P⊆NP。
- Open problem: P=NP? 美国克雷数学研究所百万大奖难题。





计算复杂性理论应用—密码学

- 私钥密码体制，加、解密规则一致，易破解
  - 明文、密钥、密文、加密算法、解密算法
- 公钥密码系统 RSA——提出者获得图灵奖
  - 利用加密秘钥→解密密钥 是一个NP Complete问题，在有效时间不可能求解，本质是大合数的真因子分解
  - 密码体系建立在NP Complete问题上，若NP Complete能在多项式时间内求解，则结果是可怕的

计算复杂性理论应用—密码学

公钥密码系统 RSA

加密密钥公开，解密密钥私有

例：三月二十八日早晨七点不发起总攻

三月二十八日早晨七点不发起总攻
解密密钥：质数位无效。
嵌入了干扰字符，可利用猜、统计、数学方法等解密。