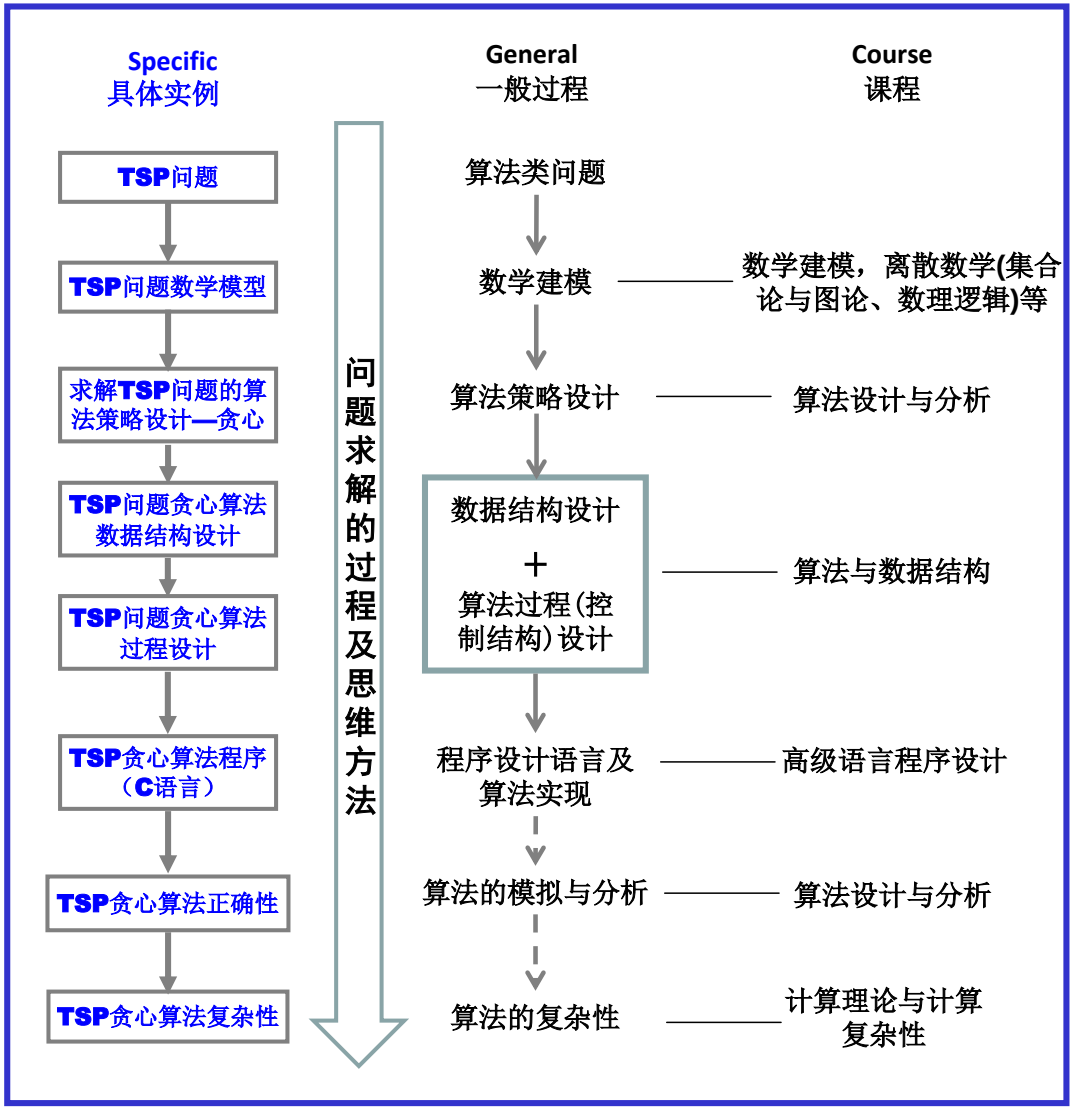
第6讲 算法：程序与计算系统之灵魂

算法是计算学科和计算系统的灵魂，各学科利用计算系统进行问题求解的关键是发现、构造与设计求解问题的算法。理解：构造与设计任何一个算法要经过哪些步骤，在每一步骤中要做哪些事情？

内容提要

基本目标: 理解算法类问题求解框架



算法与算法类问题求解

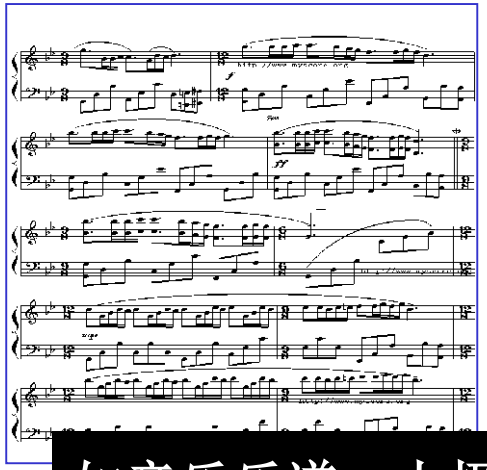
----什么是算法

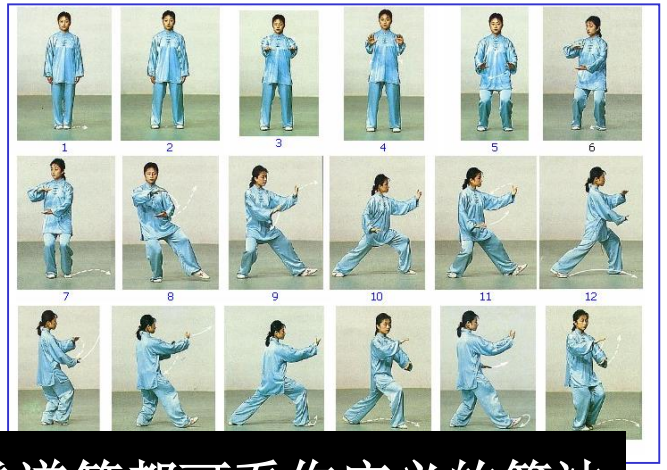
----算法类问题及求解概述

算法

◆算法---计算学科和计算机器的灵魂。“算法”(Algorithm)一词源于数学家的名字：公元825年，阿拉伯数学家阿科瓦里茨米(AlKhowarizmi)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。**

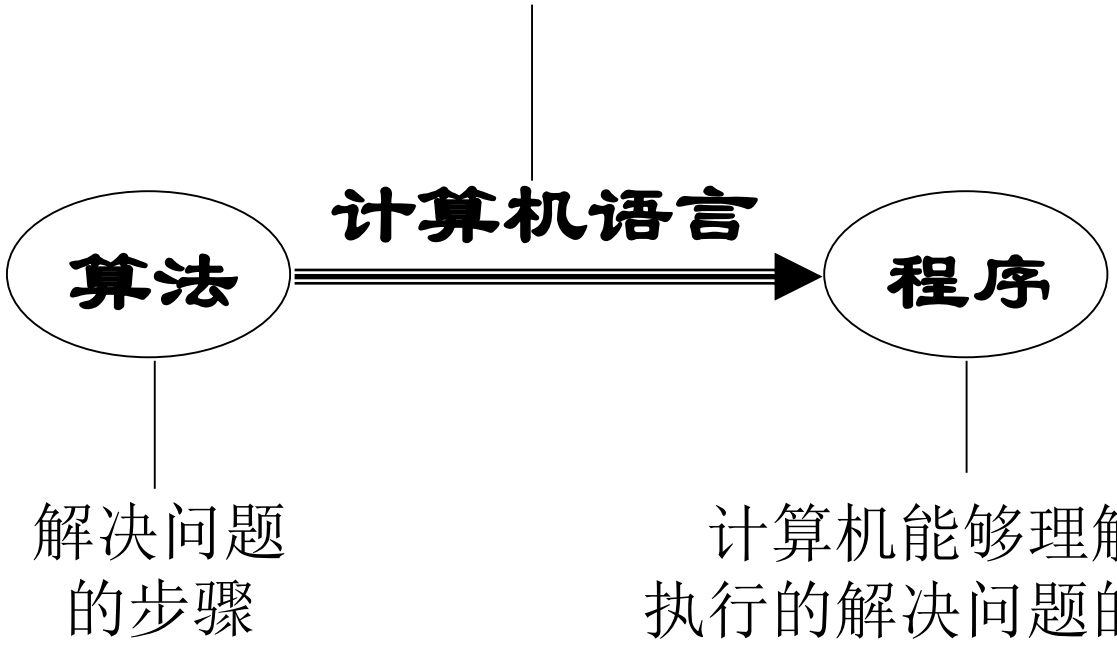




如音乐乐谱、太极拳谱等都可看作广义的算法

算法、语言与计算机程序

步骤书写的规范、语法规则、标准的集合
是人和计算机都能理解的语言



“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

算法示例

◆欧几里德算法：**求解两个数的最大公约数的算法(公元前300年)**

✓表述了最大公约数的求解过程

✓表述了一个判定过程，即判定“m和n是互质的”（即除1以外，m和n没有公约数）命题的真假。

✓该算法体现了输入、输出、有穷规则、确定性和能行性等算法的基本特征。

	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

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。

算法的基本特征

✓**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”

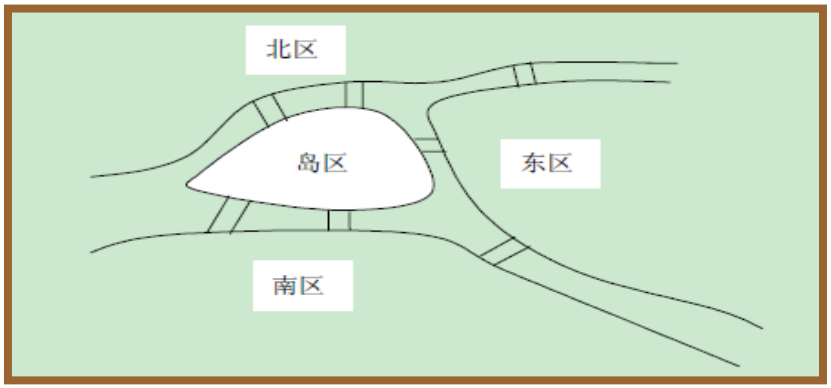
你知道一些典型的算法类问题吗?

算法类问题： 由一个算法可以解决的问题

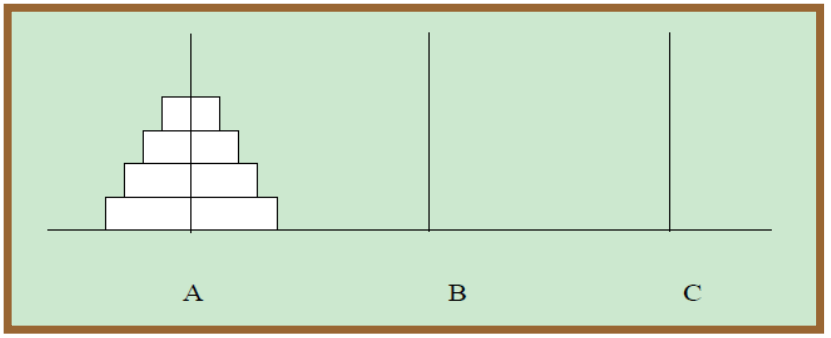
◆算法(类)问题：寻找一个(唯一的)方法(算法)以解决同一类型的无穷多个单个问题系列的问题。

◆典型问题：

✓**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”“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?”。



✓**梵天塔问题**：有三根柱子，梵天将64个直径大小不一的金盘子按照从大到小的顺序依次套放在第一根柱子上形成一座金塔，要求每次只能移动一个盘子，盘子只能在三根柱子上来回移动不能放在他处，在移动过程中三根柱子上的盘子必须始终保持大盘在下小盘在上。



✓其他如：**背包问题**，**丢番图方程可解性问题**；……

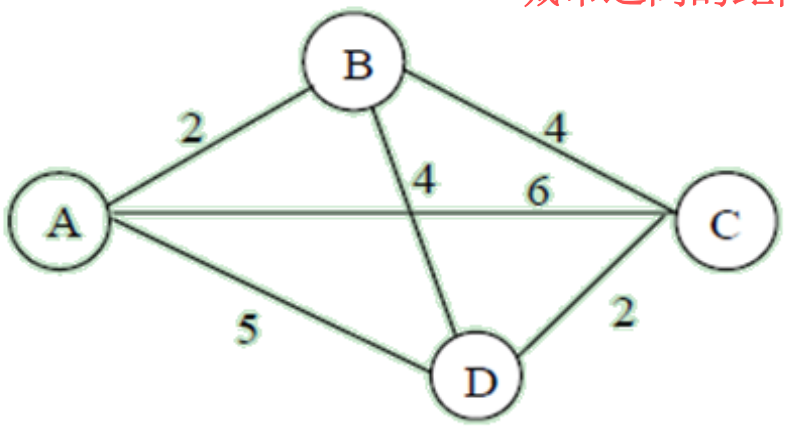
算法类问题示例

◆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。**

◆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。

城市之间的距离



算法类问题求解框架

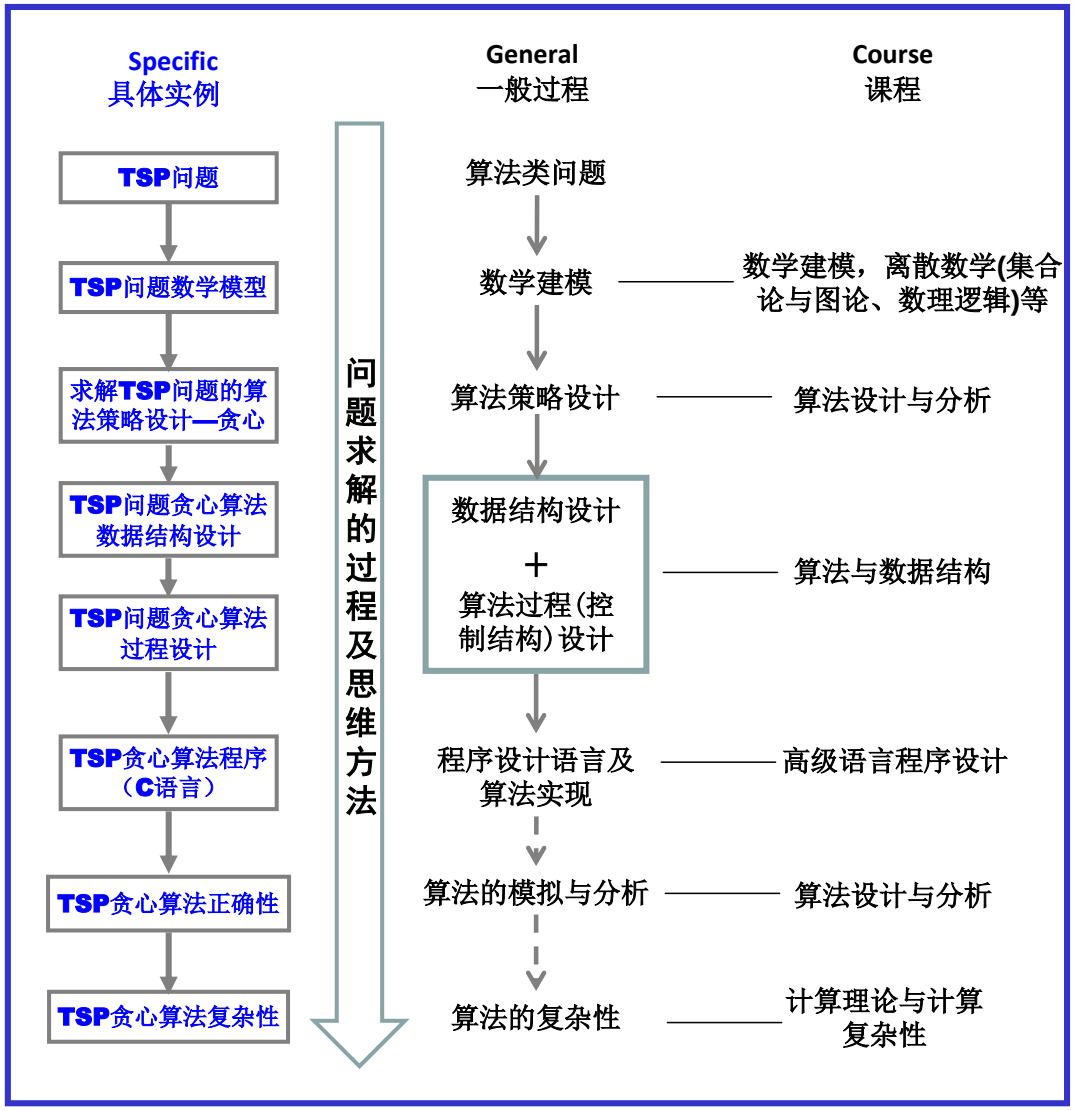
◆**问题抽象及数学建模**：将问题抽象为一个数学问题，并给出求解该数学问题的算法模型。

◆**算法策略设计**

◆**算法的数据结构和控制结构设计**：将数学模型转换为可由计算机自动计算的算法。

◆**算法的实现**：用程序设计语言编写算法实现的程序。

◆**算法的分析**：分析算法的正确性和复杂性，判断能行性！



数学建模与算法策略设计---算法思想

----数学建模

----有不同的算法求解策略

算法类问题求解的第一步就是要数学建模。

◆**数学建模：** 是一种数学的思考方法，是运用数学的语言和方法，通过抽象、简化建立对问题进行精确描述和定义的数学模型。简单而言，**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。

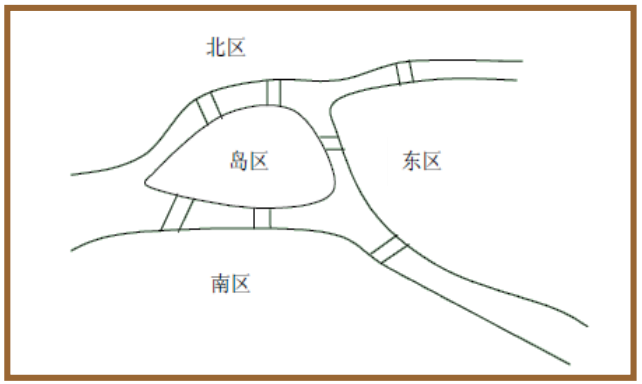
将现实世界的问题抽象成数学模型，就可能发现问题的本质及其能否求解，甚至找到求解该问题的方法和算法。

为什么说数学建模对于算法很重要?

哥尼斯堡七桥问题被抽象成一个**“图(Graph)”**

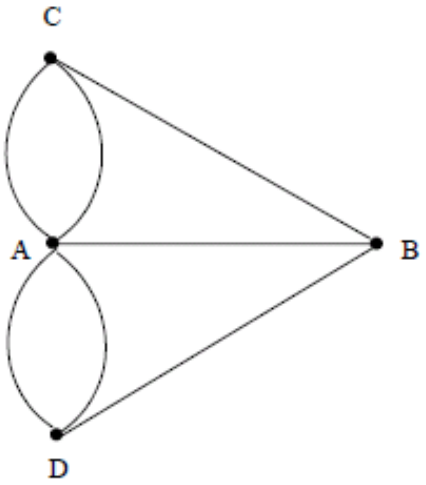
--由节点和边所构成的一种结构，

--依据“图”，可发现该问题所蕴含的许多性质：



- “**回路**---从一个节点出发最后又回到该节点的一条路径”
- “**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”
- **“经过图中每边一次且仅一次的回路”**
- “经过图中每个节点一次且仅一次的回路”
- **什么情况下有解，什么情况下无解？**

◆注：《离散数学》或者《集合论与图论》课程将介绍图的有关性质和方法。



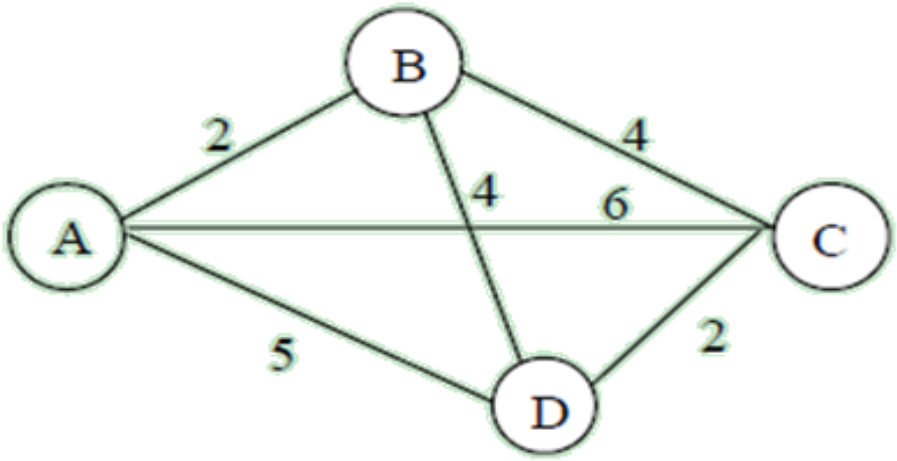
如果能抽象成数学模型，则可将一个具体问题的求解，推广为一类问题的求解！

TSP问题的数学模型

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

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

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 $\sum_{i=1}^{n} d_{t_it_{i+1}}$ 最小的访问顺序 T_{opt}。



算法策略设计---算法思想

当数学建模完成后，就要设计算法的策略或者说问题求解的策略。

◆TSP问题的基本求解策略

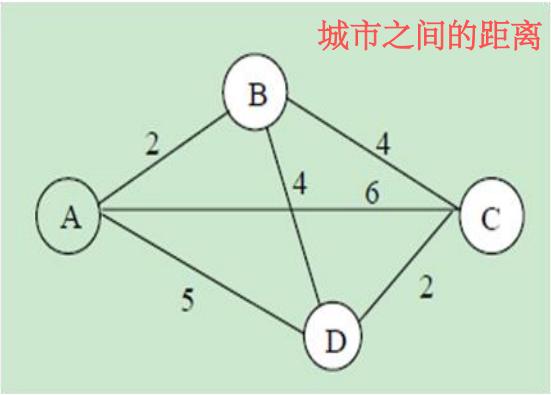
◆**遍历**：列出每一条可供选择的路线，计算出每条路线的总里程，最后从中选出一条最短的路线。

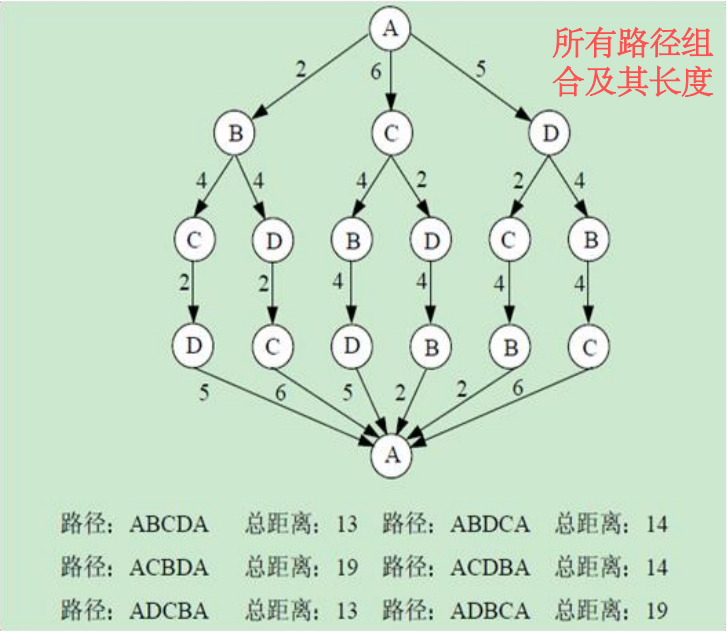
◆**出现的问题是：组合爆炸**

✓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。





◆TSP**问题的难解性**：随着城市数量的上升，TSP问题的“遍历”方法计算量剧增，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.

算法策略设计---算法思想

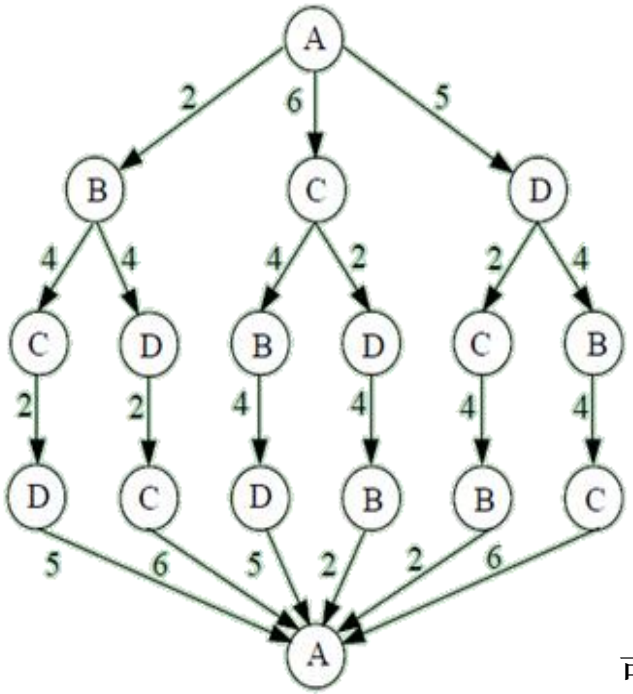
◆**TSP问题，有没有其他求解算法呢？**

◆**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
- ✓ ……

◆可选取一种合适的策略来求解TSP问题



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

贪心算法

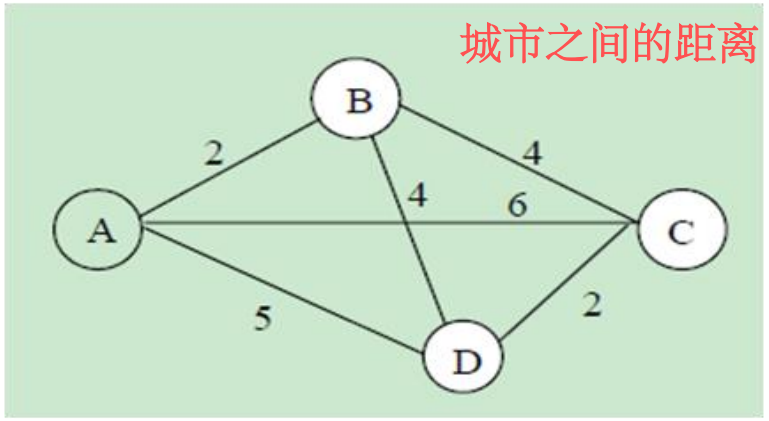
◆**贪心算法**是一种算法策略，或者说问题求解的策略。基本思想“今朝有酒今朝醉”。

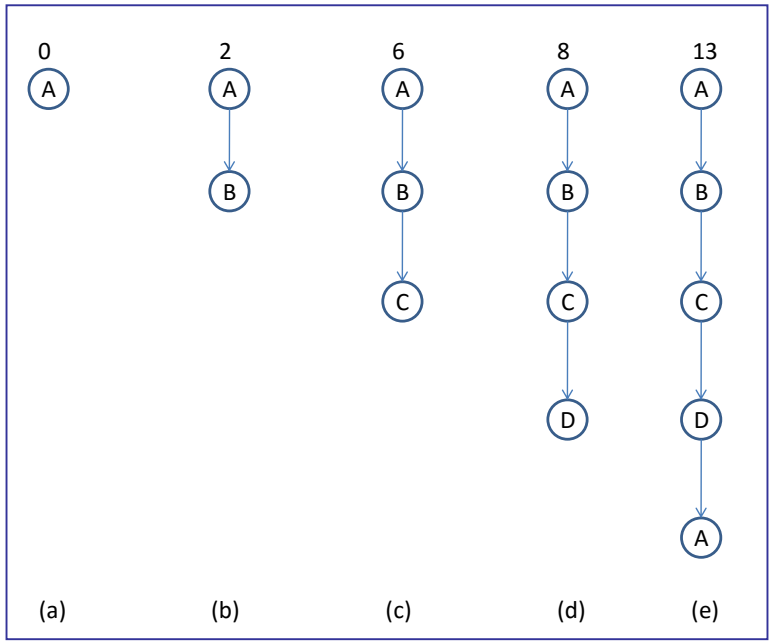
✓**一定要做当前情况下的最好选择，否则将来可能会后悔，故名“贪心”。**

◆TSP问题的贪心算法求解思想

✓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。**





算法设计---
算法思想的精确表达(I)

----算法的数据结构设计

----算法的控制结构设计及其表达方法

----TSP算法解读

算法设计

■**算法的数据结构设计**---问题或算法相关的数据之间的逻辑关系及存储关系的设计

如何将数学模型中的数据转为计算机可以存储和处理的数据？

■**算法的控制结构设计**---算法的计算规则或计算步骤设计

如何构造和表达处理的规则，以便能够按规则逐步计算出结果？

数据结构

◆如何组织、记忆、改变和操作数据的集合呢？数据存在什么结构呢？

✓**数据的逻辑结构**：数据之间的关系；数学模型中反映的通常是数据的逻辑结构。

✓**数据的存储结构**：在反映数据逻辑关系的原则下，数据在存储器中的存储方式。典型的有顺序存储结构和链式存储结构。

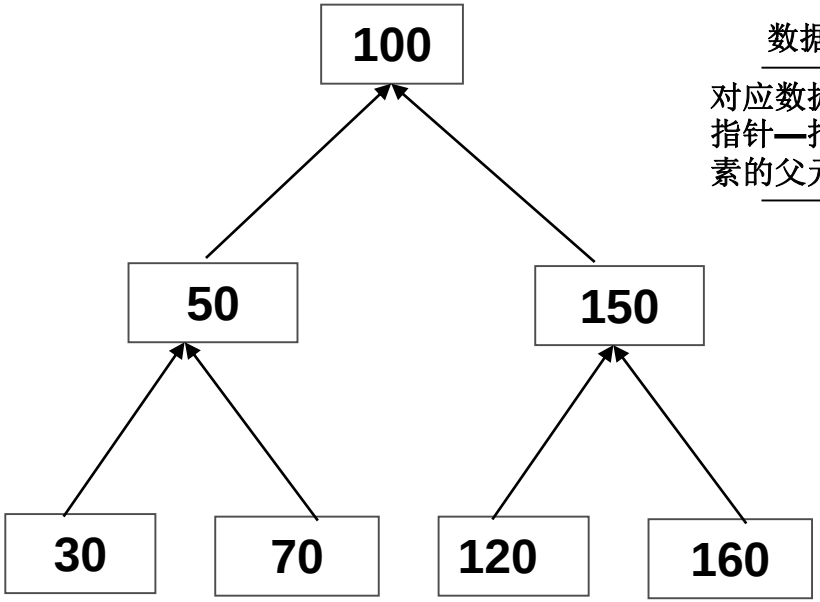
✓面向数据存储结构的**基本运算**：(1)建立数据结构；(2)清除数据结构；(3)插入数据元素；(4)删除数据元素；(5)更新数据元素；(6)查找数据元素；(7)按序重新排列；(8)判定某个数据结构是否为空，或是否已达到最大允许的容量；(9)统计数据元素的个数等。

◆**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。**

数据结构的设计

典型的数据结构---“树”示例

存储结构中,用一个指针表达数据之间的逻辑关系



数据的逻辑结构

数据的存储结构

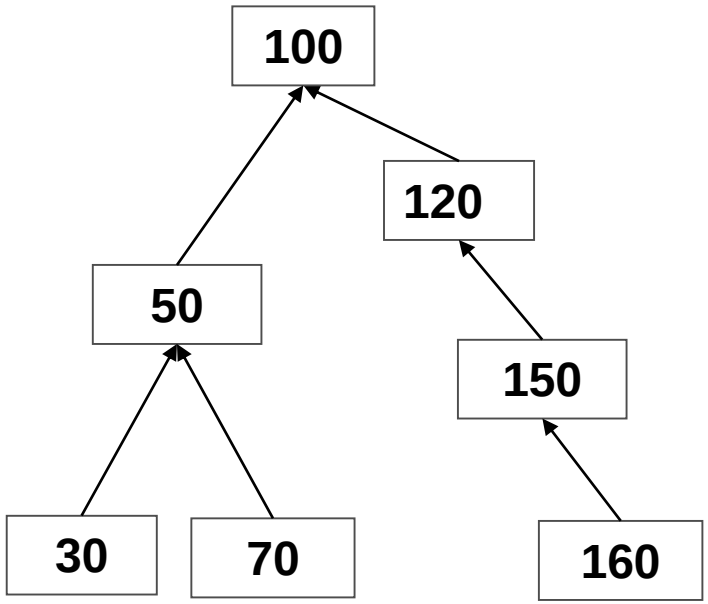
数据元素

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		

数据结构的设计

通过指针的变化, 不改变数据的存储, 但却改变了数据之间的逻辑关系



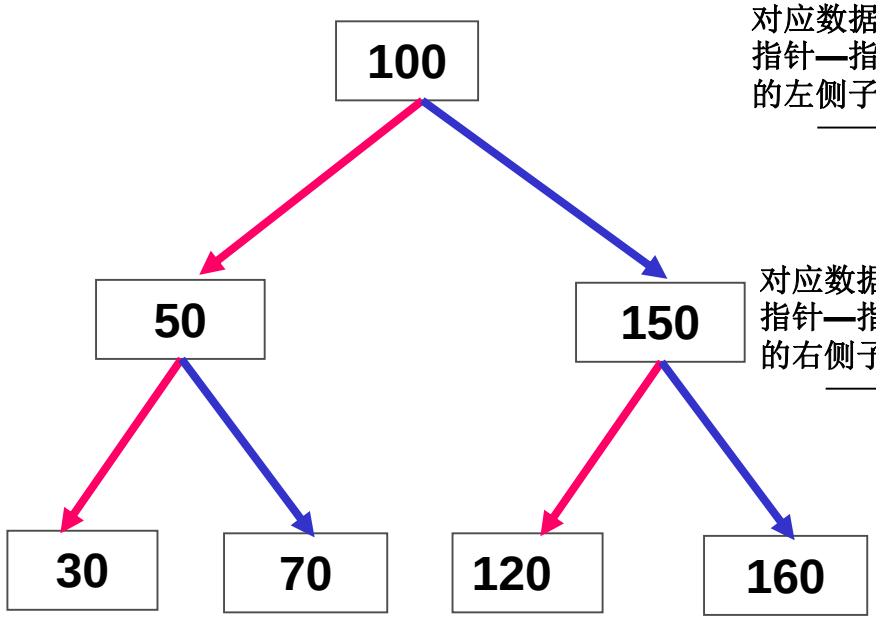
数据的逻辑结构

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		

数据结构的设计

典型的数据结构---“树”示例
另一种存储结构,用两个指针
表达数据之间的逻辑关系



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

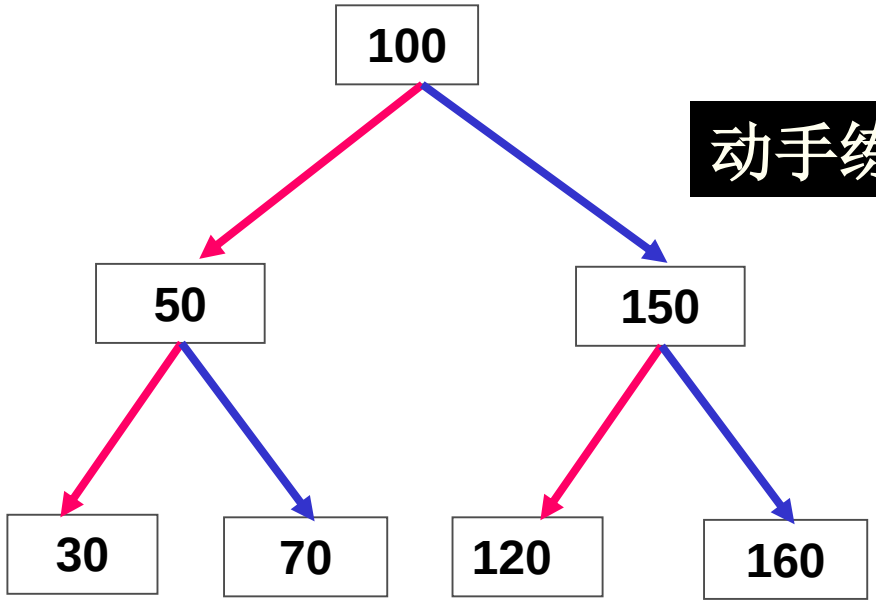
对应数据元素的右指针—指向该元素的右侧子结点

	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

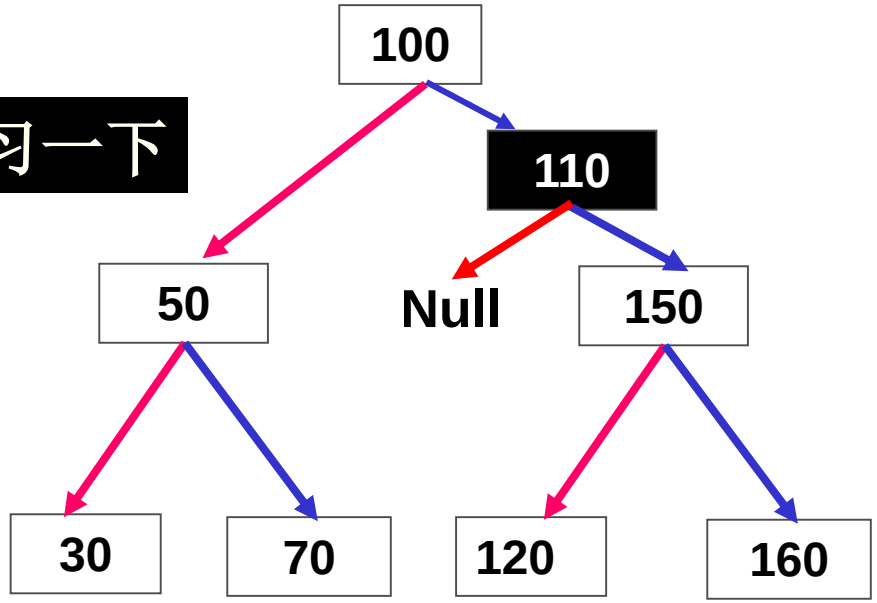
数据结构的设计

典型的数据结构--- “树”示例
另一种存储结构, 用两个指针(左指针和右指针)表达数据之间的逻辑关系



数据的逻辑结构

动手练习一下



数据的逻辑结构

多元素变量及其程序处理(前讲介绍的)

- 向量或列表或数组。
- 矩阵或表

向量实例

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

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

表实例

列

行	1	2	3	4
1	11	25	22	25
2	45	39	8	44
3	21	28	0	100
4	34	83	75	16

M[2,3]

```
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;
```

TSP问题的数据结构设计

◆数据结构设计就是针对选定的算法策略，设计其相应的数据结构及其运算规则。不同的算法可能有不同的数据结构及其运算规则！

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}

算法设计---
算法思想的精确表达(II)

----算法的数据结构设计

----算法的控制结构设计及其表达方法

----TSP算法解读

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。

算法与程序构造的表达方法：程序流程图

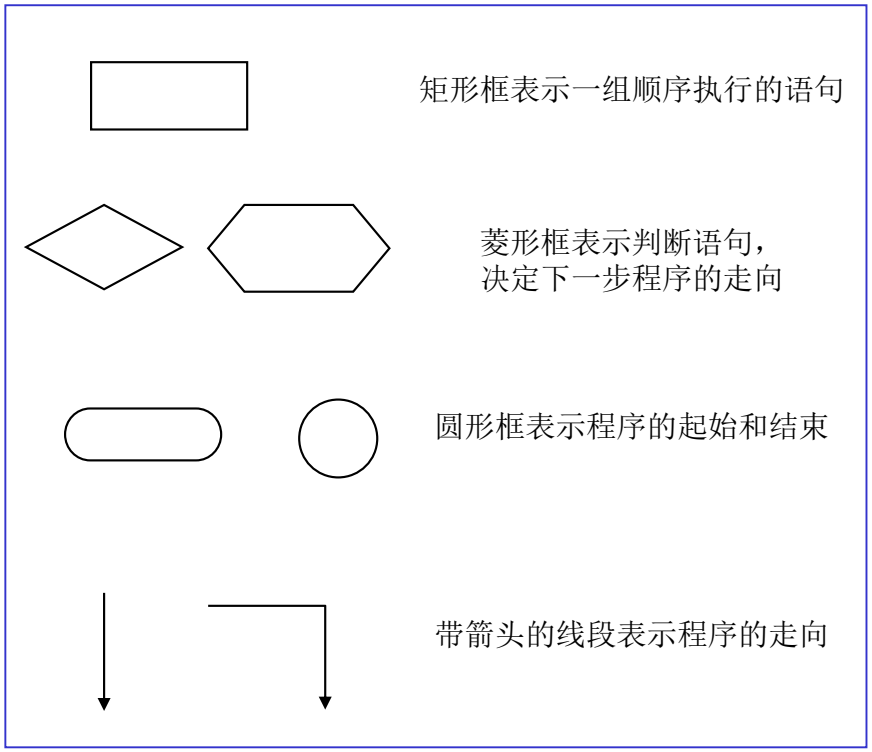
流程图的基本表示符号

✓**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。

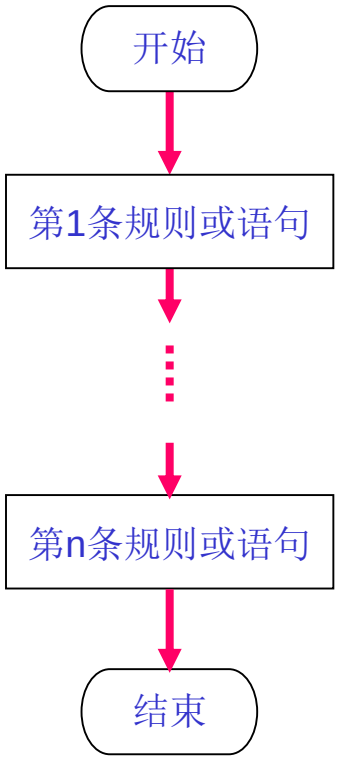
✓**圆形框**：表示算法或程序的开始或结束。

✓**箭头线**：表示算法或程序的走向。

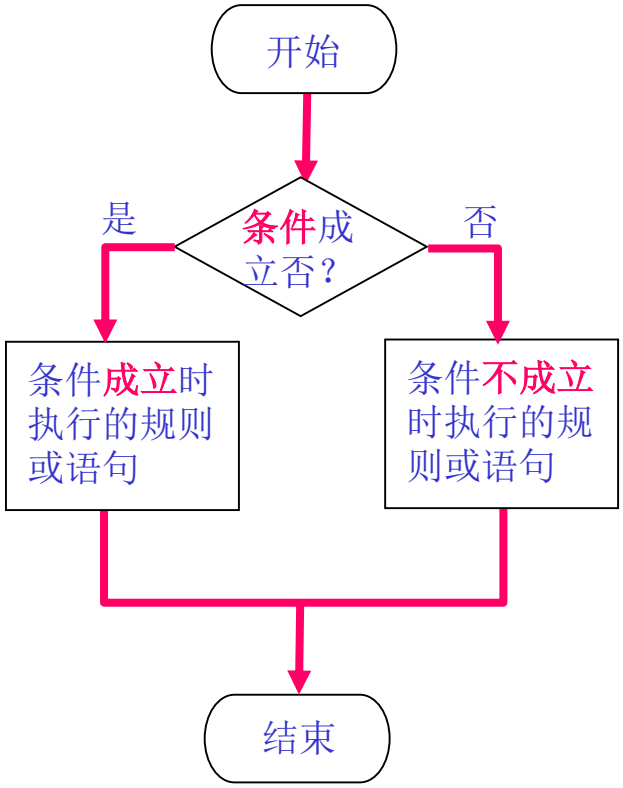


算法与程序构造的表达方法：程序流程图

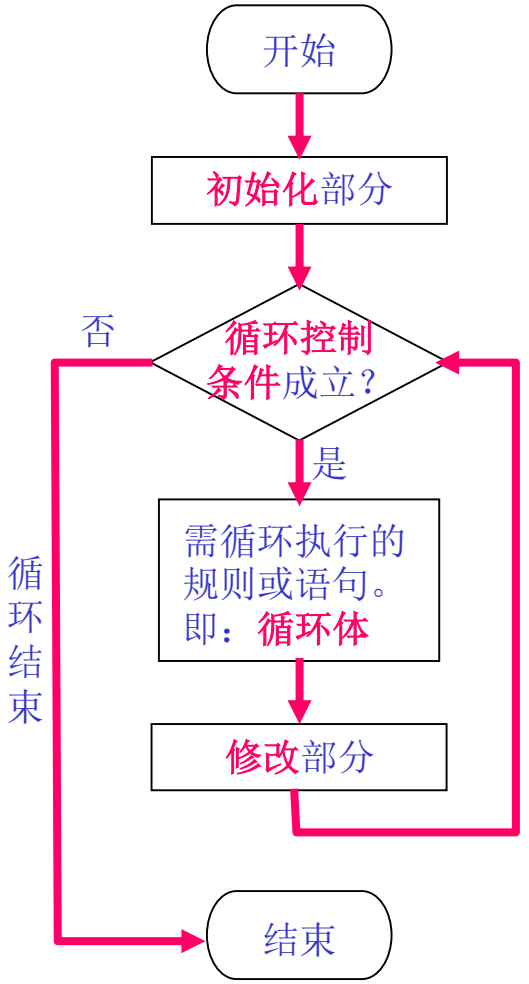
◆三种控制结构的流程图表示方法示意



顺序结构的流程图



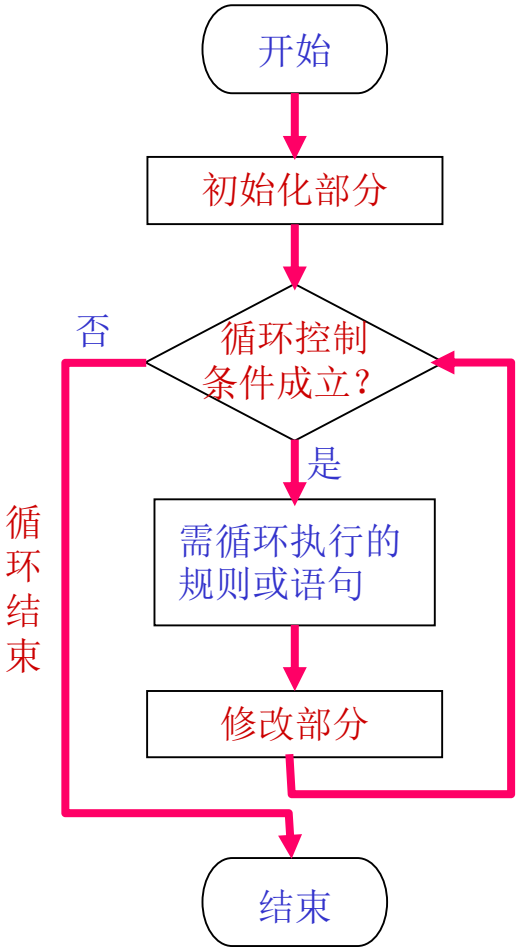
分支结构的流程图



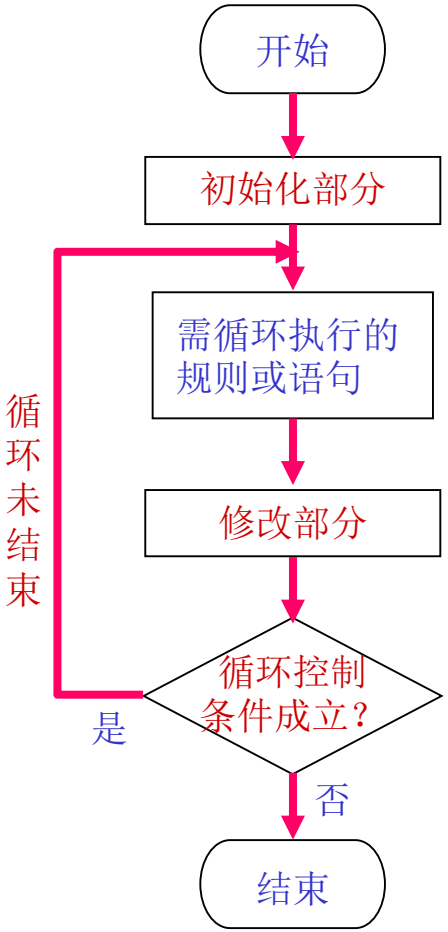
循环结构的流程图

算法与程序构造的表达方法：程序流程图

◆循环结构的两种情况的流程图表示方法示意



有界循环结构的流程图(循环次数确定)

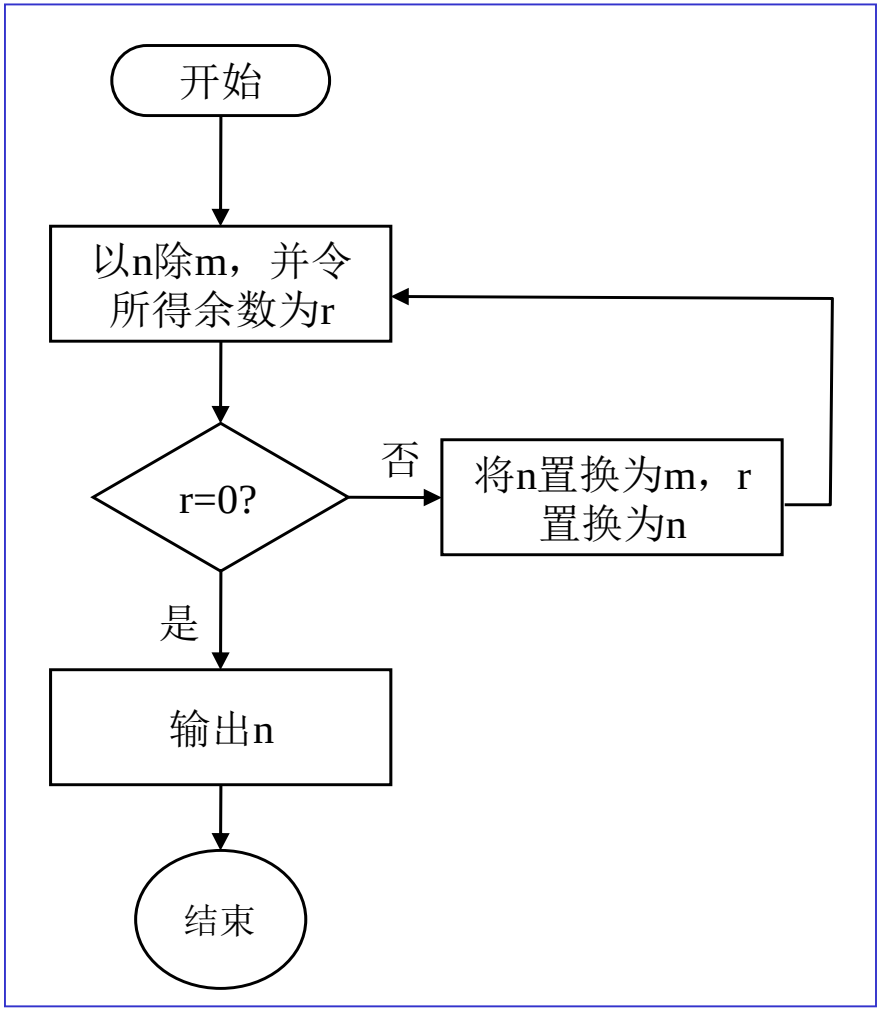


条件循环结构的流程图(循环次数不确定)

算法与程序构造的表达方法：程序流程图

◆算法思想及算法流程图表示示例

◆欧几里德算法流程图



算法与程序构造的表达方法：步骤描述法

◆步骤描述法，即用人们日常使用的语言和数学语言描述算法的步骤。

◆例如：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

算法设计---
算法思想的精确表达(III)

----算法的数据结构设计

----算法的控制结构设计及其表达方法

----TSP算法解读

求解TSP问题的遍历算法

✓遍历所有的组合路径;

✓累加一条路径的距离之和;

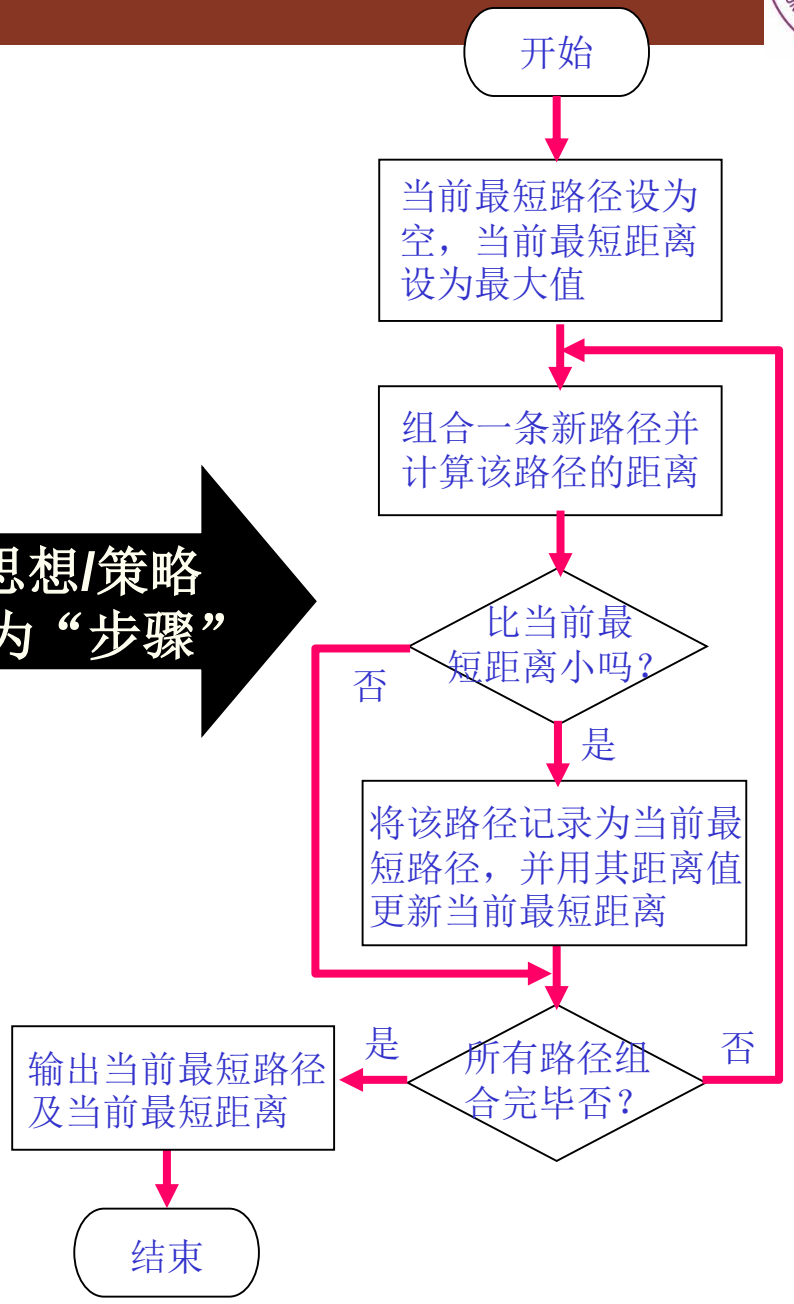
✓判断某条路径的距离是不是比当前最短路径距离短;

✓如果是，则用新路径取代当前最短路径，并记录其距离;

✓直到所有路径比较完毕;

◆……

将思想/策略转变为“步骤”



步骤描述法表示的求解

TSP问题的贪心算法

✓城市用数字编号来表示,1,2…,N

✓任何两个城市的距离记录在数组D[i,j]中

✓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)：从第1个城市开始访问起，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次访问城市的距离最短。

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);
(11)Sum=Sum+D[1, j];
(12)逐个输出S[N]中的全部元素;
(13)输出Sum。

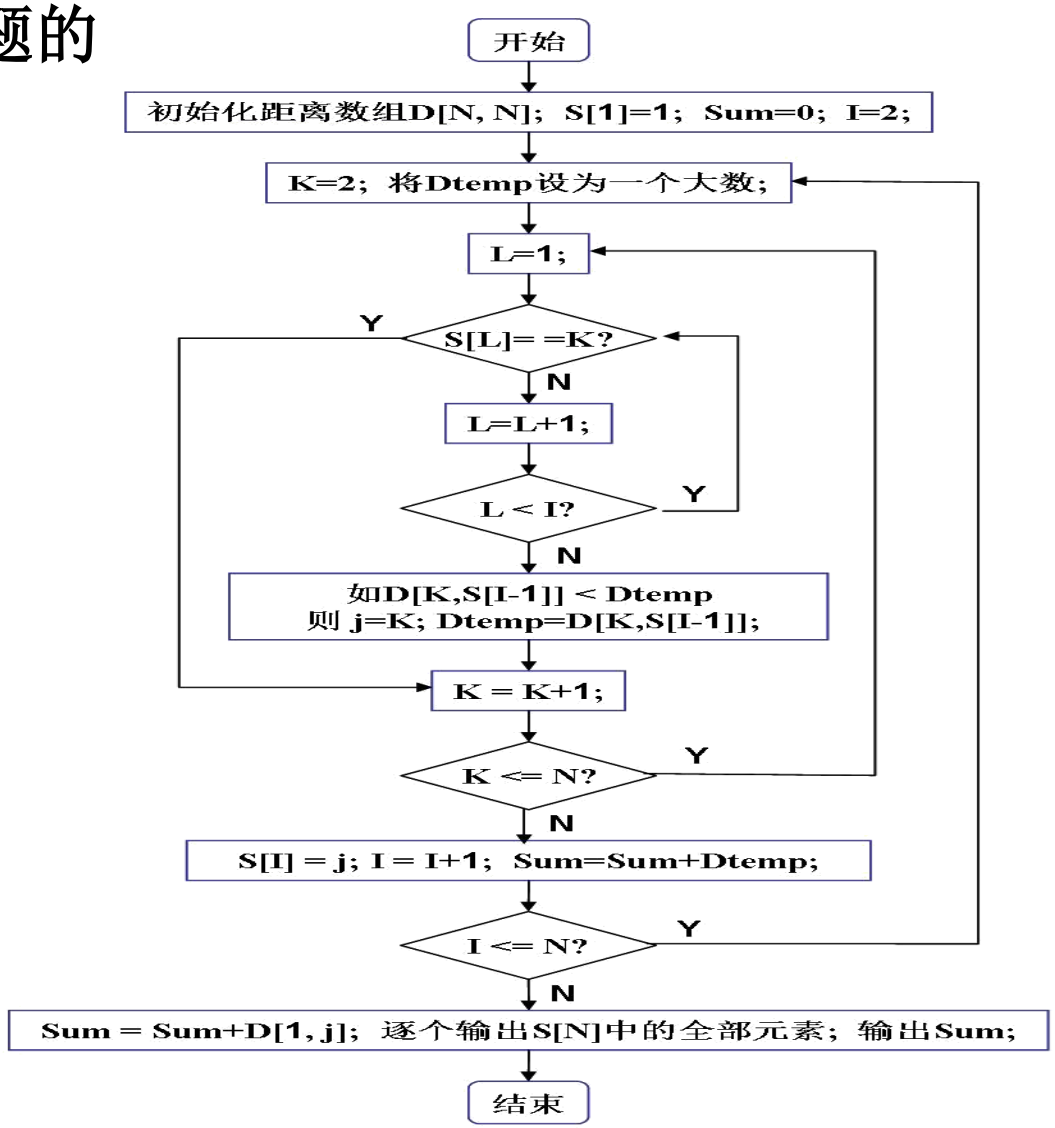
End of the Algorithm

步骤描述法表示的求解TSP问题的贪心算法(Cont.)

◆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

(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; j=K; Dtemp＝D[K,S[I-1]];
(5.8)K=K+1;
(5.9)如果K<=N，转步骤5.3。

流程图表示的求解TSP问题的贪心算法



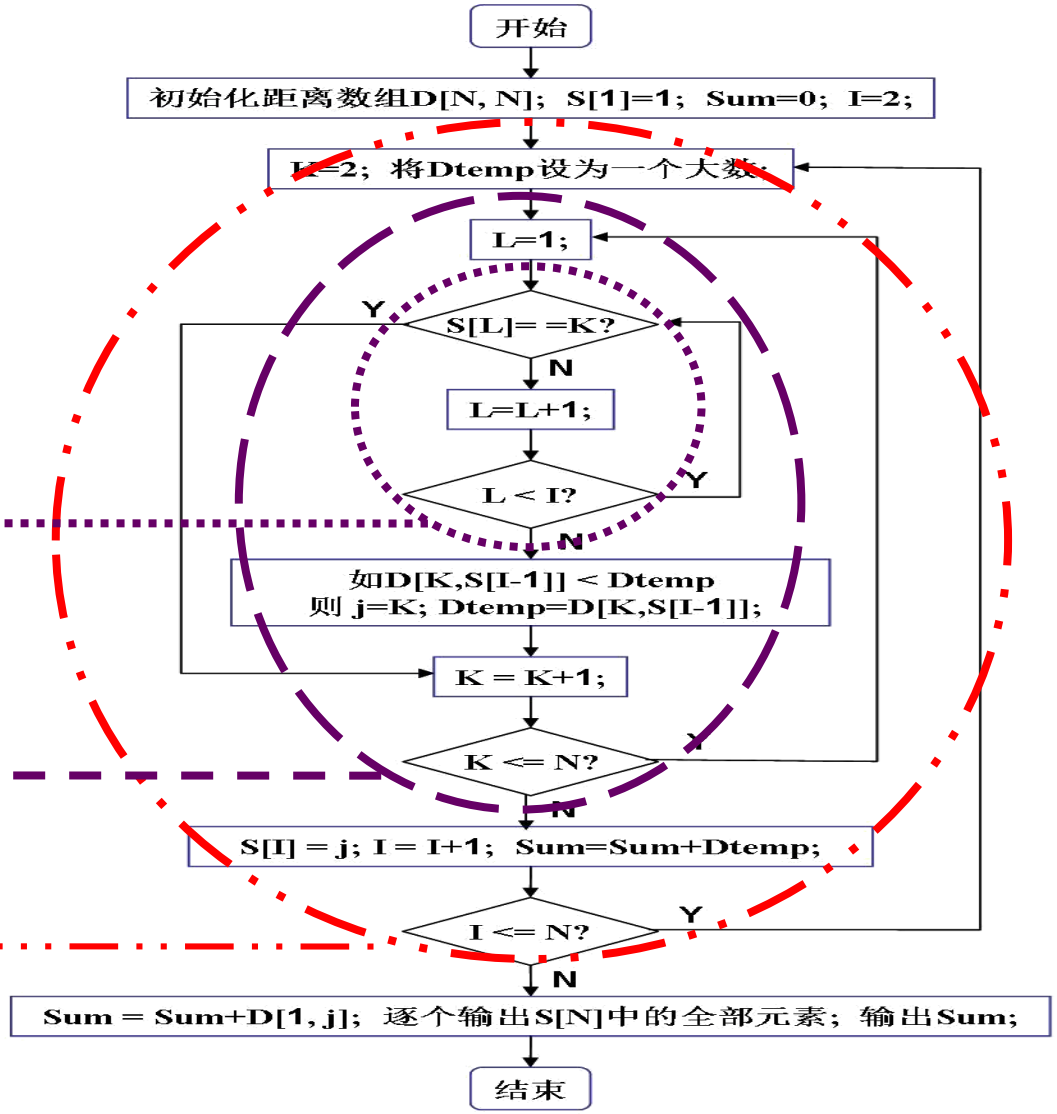
算法思想解读

◆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个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从第2个城市至第N个城市循环, 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。

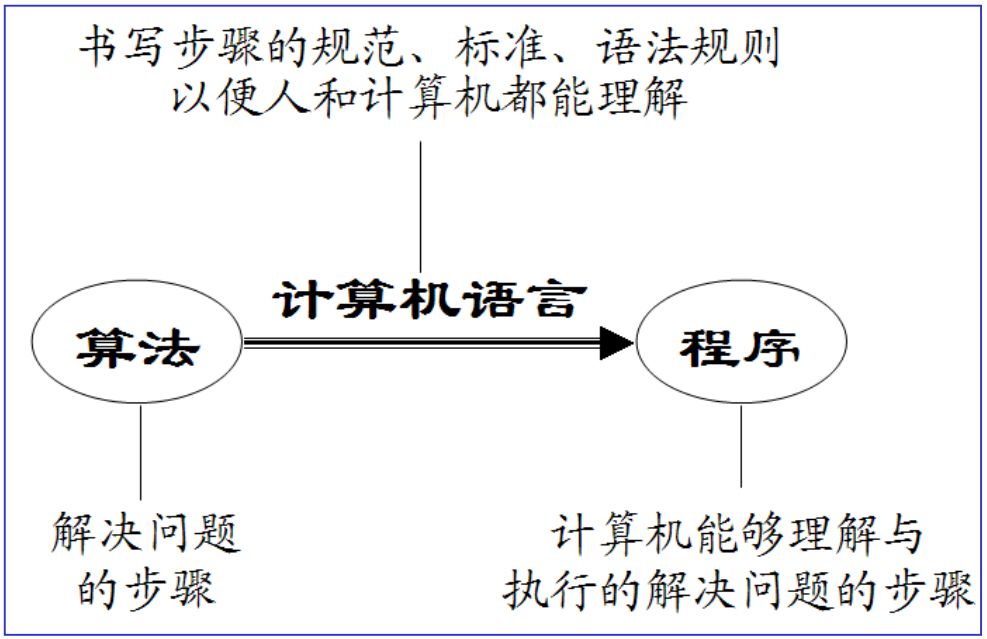


算法实现---程序设计

算法的实现

◆**程序**是算法的一种机器相容(Compatible)的表示，是利用计算机程序设计语言对算法描述的结果，是可以在计算机上执行的算法。

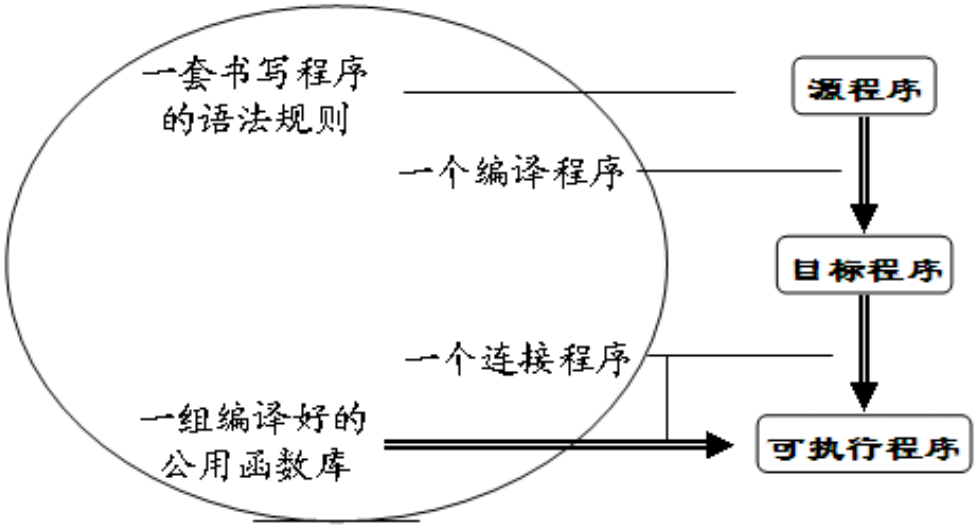
◆**计算机语言**是书写算法步骤地规范、标准、语法规则等，以便使人和计算机能够理解用计算机语言编写出的程序(注：更重要的是使计算机能够理解)。



算法的实现

◆**程序设计过程**：一般经过编辑源程序➔编译➔链接➔执行。所谓编辑源程序是利用程序编辑器，按照计算机语言的规范书写程序的过程；所谓编译是将编制好的源程序翻译成机器可以执行的机器代码程序(又称为目标代码)的过程。所谓链接是将目标代码与公用函数库中的函数实现代码集成起来形成最终可执行程序的过程。所谓执行就是程序的运行过程。

◆**计算机语言程序设计环境**通常由一套书写程序的语法规则、一个编译程序、一个链接程序和一组编译好的公用函数库构成。有些计算机语言同时也提供了相应的编程环境、调试环境及集成开发环境等。



环境：编辑、编译、连接、调试、运行一体化平台

算法的实现

TSP 问 题 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---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
}
```

高级问题初探:
算法分析与计算复杂性

算法分析与计算复杂性

算法是正确的吗?

◆算法的模拟与分析

◆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？
- ✓ 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?

算法分析与计算复杂性

算法分析示例：TSP问题贪心算法的模拟与分析

◆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

◆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。

算法分析与计算复杂性

算法获得结果的时间有多长?

◆另一个问题：**算法是能够执行的吗**？

◆**算法的复杂性分析**

◆**算法的效率**：时间效率和空间效率

◆**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，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$n^2+n+1 =O(n^2)$ 。

◆**算法的空间复杂度**：算法在执行过程中所占存储空间的大小。

算法分析与计算复杂性

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

算法分析与计算复杂性

算法复杂性分析示例：TSP问题算法的复杂性

◆TSP问题**遍历算法**的复杂性：

✓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

✓组合爆炸：路径组合数目为(n-1)!

✓时间复杂度是**O((n-1)!)**

◆TSP问题**贪心算法**的复杂性：

✓粗略看是一个关于n的三重循环，即复杂度为n^3级别。

✓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+D[I][j];
  } while(I<n);           //I 从 1 到 n-1 循环
  Sum=Sum+D[I][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

算法分析与计算复杂性

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

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

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

$20^3 = 8000$

$O(n^3)$	$O(3^n)$
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

算法分析与计算复杂性

◆难解性问题

◆当算法的时间复杂度的表示函数是一个多项式时，如$O(n^2)$时，则计算机对于大规模问题是可以处理的。例如，TSP问题的贪心算法 $O(n^3)$ 。

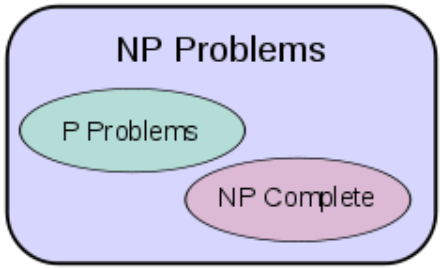
◆当算法的时间复杂度是用指数函数表示时，如$O(2^n)$，当n很大（如10000）时计算机是无法处理的，在计算复杂性中将这一类问题被称为**难解性问题**。例如，TSP问题的遍历算法。

◆计算复杂性理论

◆所有可以在多项式时间内求解的问题称为P类问题

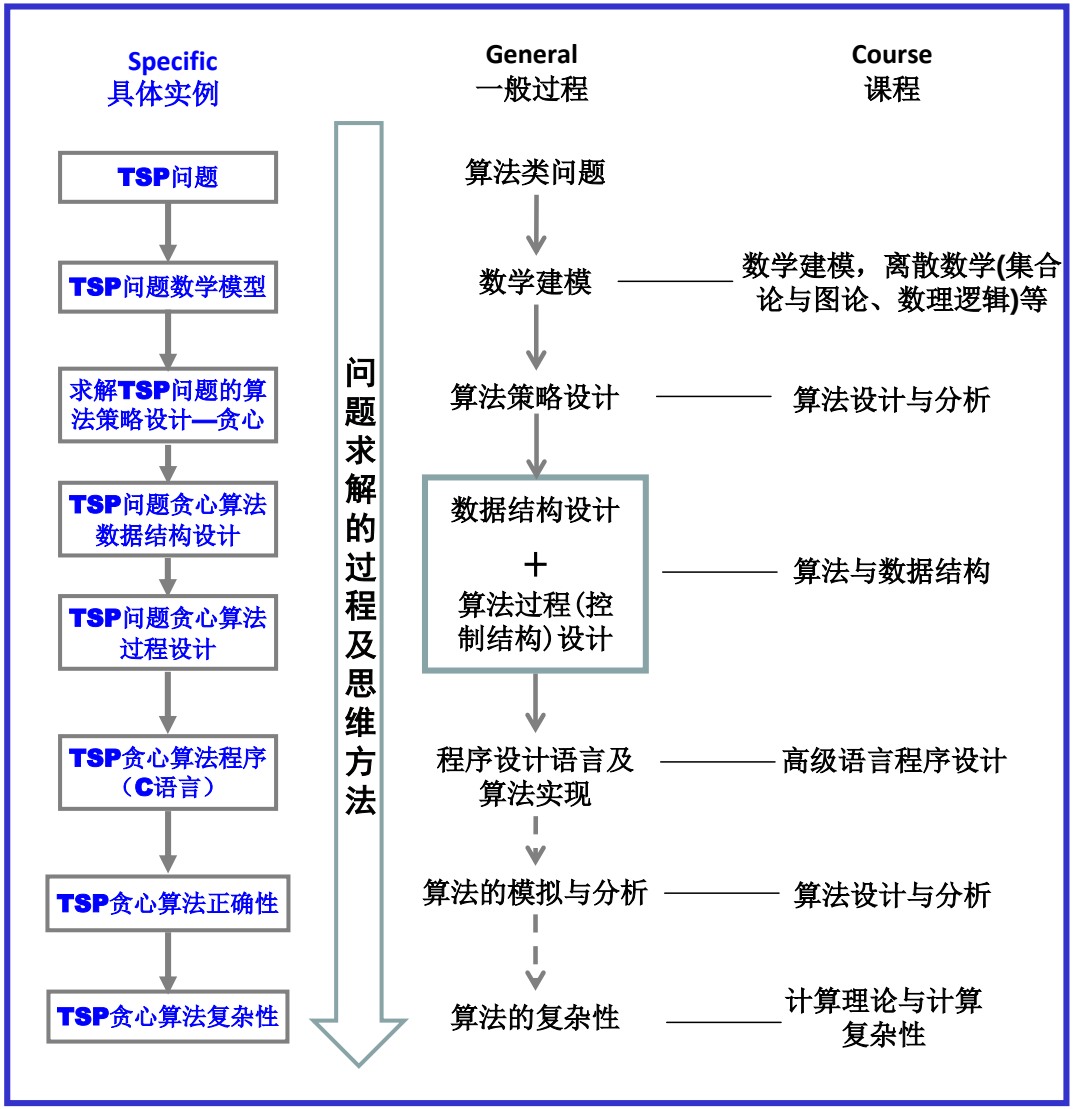
◆所有在多项式时间内可以验证的问题称为NP类问题，P⊆NP。

◆Open problem: P=NP? 美国克雷数学研究所百万大奖难题。



本讲小结

基本目标: 理解算法类问题求解框架



第6讲 算法：程序与计算系统之灵魂

Questions & Discussion?

2015