《人工智能》课后习题答案

第一章　绪论

1.1 答：人工智能就是让机器完成那些如果由人来做则需要智能的事情的科学。人工智能是相对于人的自然智能而言，即用人工的方法和技术，研制智能机器或智能系统来模仿延伸和扩展人的智能，实现智能行为和“机器思维”，解决需要人类专家才能处理的问题。

1.2 答：“智能”一词源于拉丁“***Legere***”，意思是收集、汇集，智能通常用来表示从中进行选择、理解和感觉。所谓自然智能就是人类和一些动物所具有的智力和行为能力。

智力是针对具体情况的，根据不同的情况有不同的含义。“智力”是指学会某种技能的能力，而不是指技能本身。

1.3 答：专家系统是一个智能的计算机程序，他运用知识和推理步骤来解决只有专家才能解决的复杂问题。即任何解题能力达到了同领域人类专家水平的计算机程序度可以称为专家系统。

1.4 答：

自然语言处理—语言翻译系统，金山词霸系列

机器人—足球机器人

模式识别—Microsoft Cartoon Maker

博弈—围棋和跳棋

第二章 知识表达技术

2.1 解答：

（1）状态空间(State Space)是利用状态变量和操作符号，表示系统或问题的有关知识的符号体系，状态空间是一个四元组（S，O，S0，G）：

S—状态集合;O—操作算子集合;S0—初始状态,S0⊂S;G—目的状 态,G⊂S,(G 可若干具体状态，也可满足某些性质的路径信息描述)

从 S0 结点到 G 结点的路径被称为求解路径。

状态空间一解是一有限操作算子序列，它使初始状态转换为目标状态：

$$S0 \xrightarrow{O1} S1 \xrightarrow{O2} S2 \xrightarrow{O3} \cdots\cdots \xrightarrow{Ok} G$$

其中 O1，…，Ok 即为状态空间的一个解(解往往不是唯一的)

（2）谓词逻辑是命题逻辑的扩充和发展，它将原子命题分解成客体和谓词两个部分。

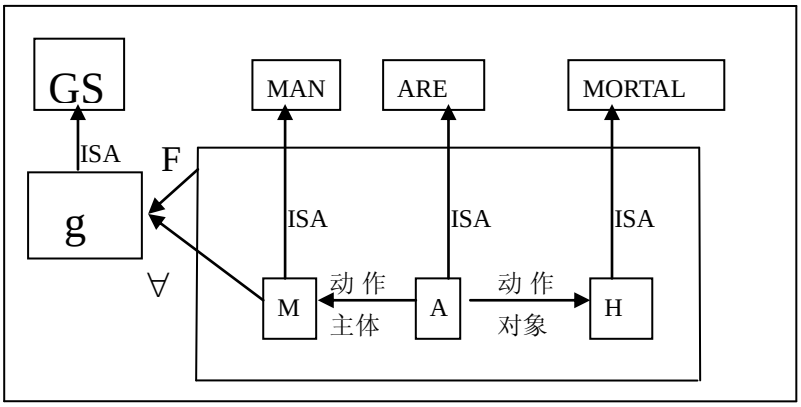
与命题逻辑中命题公式相对应，谓词逻辑中也有谓词（命题函数）公式、原子谓词公式、复合谓词公式等概念。一阶谓词逻辑是谓词逻辑中最直观的一种逻辑。

（3）语义网络是一种采用网络形式表示人类知识的方法。即用一个有向图表示概念和概念之间的关系，其中节点代表概念，节点之间的连接弧(也称联想弧)代表概念之间的关系。

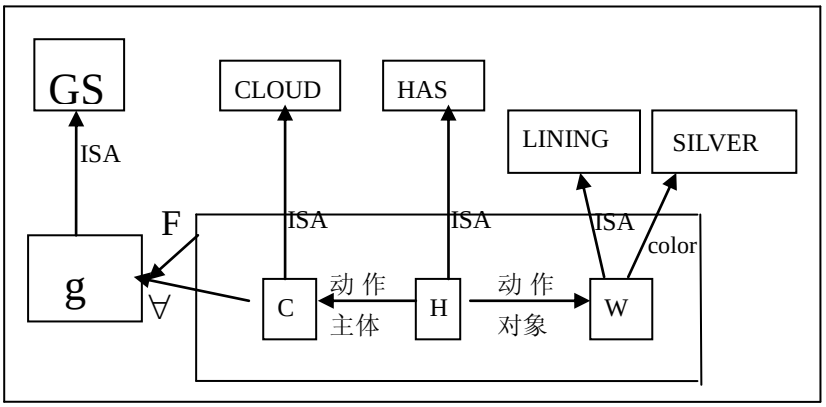
常见的语义网络形式有命题语义网络、数据语义网络：E-R 图（实体-关系图）、语言语义网络等。

2.2 解答：

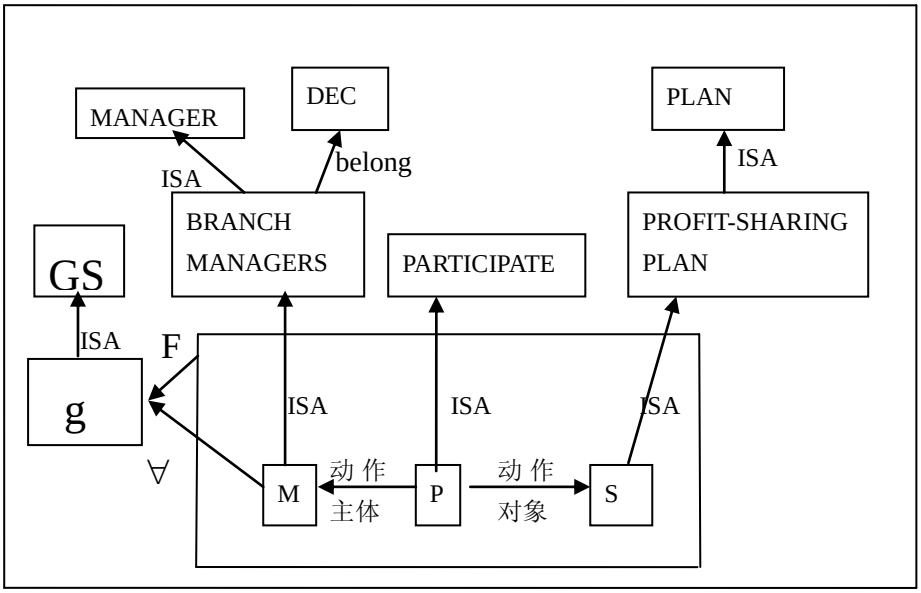
（1）


GS
MAN
ARE
MORTAL
ISA
F
g
ISA
ISA
ISA
∀
M
动 作
主体
A
动 作
对象
H

（2）


GS
CLOUD
HAS
LINING
SILVER
ISA
F
ISA
ISA
ISA
color
g
∀
C
动 作
主体
H
动 作
对象
W

(3)


MANAGER
DEC
PLAN
ISA
belong
ISA
BRANCH
MANAGERS
PARTICIPATE
PROFIT-SHARING
PLAN
GS
ISA
F
g
ISA
ISA
ISA
∀
M
动 作
主体
P
动 作
对象
S

2.3 解答：设有如下四个谓词：

HUMAN(X)　X 是人

LAWED(X)　X 受法律管制

COMMIT(X)　X 犯法

PUNISHED(X) X 受法律制裁

前两个谓词可以变为：HUMAN(X) ⟶ LAWED(X)，表示：人人都要受法律的管制；

后两个谓词可以变为：COMMIT(X)⟶PUNISHED(X)，表示只要 X 犯了罪，X 就要受到惩罚；

进一步，还可以把上述两个谓词联结成如下形式：

[HUMAN(X) ⟶ LAWED(X)] ⟶ [COMMIT(X) ⟶PUNISHED(X)]

本公式的含义是：如果由于某个 X 是人而受到法律管制，则这个人犯了罪就一定要受到惩罚。

晁盖是人，受法律的管制（老百姓受法律的管制）；所以晁盖劫了生辰纲，违反了宋王朝的法律，一定要受到官府的追究。

高衙内是人，却不受法律的管制（达官贵人和恶少不受法律的管制）；所以高衙内强抢民女，同样是违反了宋王朝的法律，却可以横行无忌。

2.4 解答：题中提供的条件可记为①②③④⑤，依次利用这些条件可得到如下结果：

（1）条件②：周和钱是同一性别；
　　条件⑤：李、徐、周是同一性别；
} 推得：李、徐、周、钱是同一性别

　　条件③：李的爱人是陈的爱人的表哥，则李的爱人性别是男，而李的性别是女

这样可以初步推出：李、徐、周、钱均是女的，对应的王、陈、孙、吴均是男的。

（2）条件④：陈与徐、周俊不构成夫妻，则陈选择的余地为钱或李；
　　条件③：李与陈不构成夫妻；
} 推得：陈与钱是夫妻

　　条件④：吴与徐、周均不构成夫妻，则吴选择的余地为李；推得：吴与李是夫妻

　　条件①：王与周不构成夫妻，则王选择的余地为徐；推得：王与徐是夫妻

排除上述已经成立的条件，显然可推得：孙与周是夫妻。

2.5 解答：符号微积分基本公式为 $\int_a^b f(x) = F(b) - F(a) = F(x)|_a^b$

用产生式表示为：If f(x) and (a,b) Then F(b)-F(a)

2.6 解答：题中描述的情况用谓词形式可表达如下：

DOG(X)　X 是狗

SOUND(X)　X 会吠叫

BIT(X,Y)　X 咬 Y

ANIMAL(X) X 是动物

题中各条推理则可以表示为：

P1: ∀X DOG(X) ⟶ ∃yBIT(X,Y）∨SOUND(X)

P2: :∀X（ANIMAL(X) ∧SOUND(X)） ⟶∃yBIT(X,Y）

P3: 猎犬是狗，即 DOG(X)种 X 的谓词样品是猎犬，同时也可得 ANIMAL(猎犬)

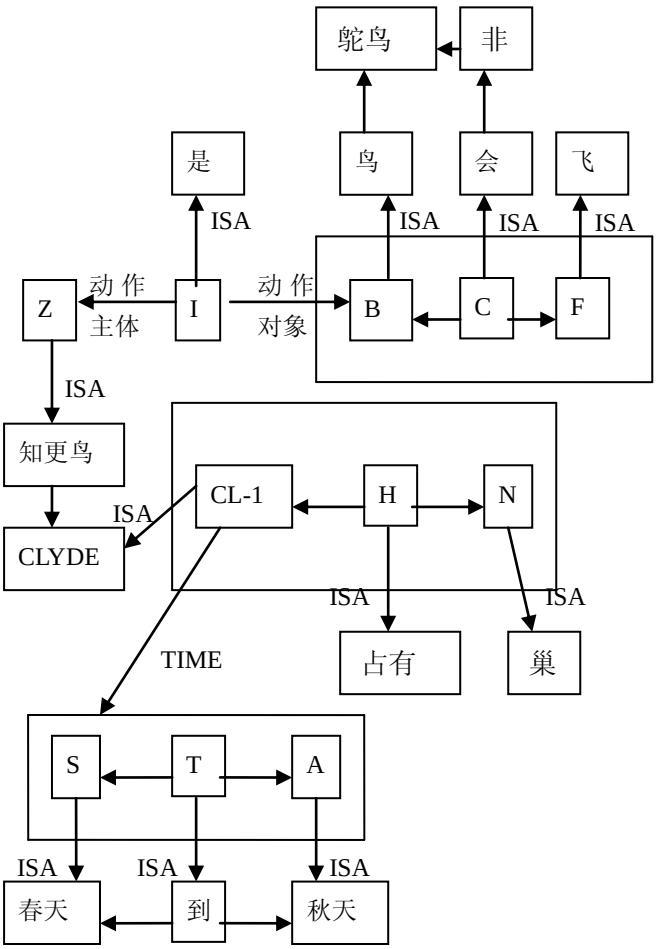
将 P3 带入 P1 可得 SOUND(猎犬)，再将 SOUND(猎犬)和 ANIMAL(猎犬)带入 P2 可得

∃yBIT(猎犬,Y)，即可以得到结果：猎犬是咬人的。

2.7 解答：题中的三条规则侧重点不同：R1 规则的重点在于我师的任务；R2 规则的重点在于敌团的配置；R3 规则的重点在于我师的任务和敌团的配置同时满足。它们之间的关系为 R1⊂ R2⊂ R3。

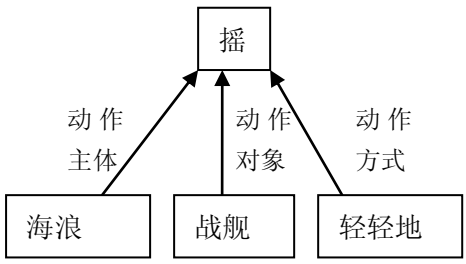
所以根据冲突解决规则中的规模排序，可知首先应该选择规则 R3，系统执行才最有效。

2.8 解答：



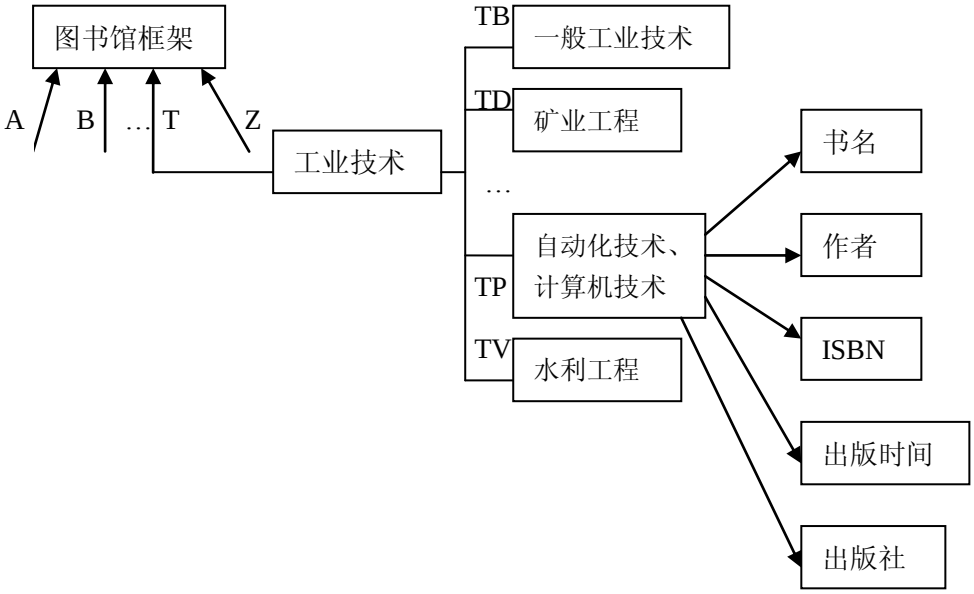
2.9 解答：

（1）



(2)

2.10 解答：



2.11 解答：

在产生式系统中，随着产生式规则的数量的增加，系统设计者难以理解规则间的相互作用，究其原因，在于每条规则的自含性使得知识表示的力度过于细微。因此要提高产生式系统的可理解性，就应当按照软件工程的思想，通过对规则的适当划分，将规则组织诚易于管理的功能模块。由于框架系统具有组织成块知识的良好特性，因此将两者进行有机结合，可以为产生式系统的开发、调试和管理提供有益的帮助。

基于框架的表示机制可以用作产生式语言和推理机制设计的一个重要构件。另外，框架可以直接用于表示规则，如果将每一个规则作为一个框架处理，一组用于解决特定问题的规则可组织成一类，且在这一类框架中表示这组规则的各种特性。

2.12 解答：略

2.13 解答：

（1）题目描述可转换为如下问题（N 阶汉诺塔问题）

有编号为 A、B、C 的三个柱子和标识为 1、2、…、N 的尺寸依次从小到大的 N 个有中心孔的金片；初始状态下 N 个金片按 1、2、…、N 顺序堆放在 A 号柱子上，目标状态下 N 个金片以同样次序顺序堆放在 B 号柱子上，金片的搬移须遵守以下规则：每次只能搬一个金片，且较大金片不能压放在较小金片之上，可以借助于 C 针。

（2）假设基本操作为 move(x,A,C,B)，表示将 x 个金片从 A 移到 B 上，中间可借助于 C。

当 N=1 时，则无需借助中间的 C 针，就可以直接实现将 1 个金片从 A 移到 B 上，这也是问题的最简操作，可表示为 move-one(1,A,B)；

当 N>1 时，需要用中间的 C 针作辅助。其操作又可分为以下三步：

将 N-1 个金片从 A 移到 C 上，中间可借助于 B，转换为基本操作就是 move(N-1,A, B, C)；

将 1 个金片直接从 A 移到 B 上，转换为基本操作就是 move-one(1,A,B)；

将 N-1 个金片从 C 移到 B 上，中间可借助于 A，转换为基本操作就是 move(N-1, C, A, B)；

这样，就将问题的规模减小为 N-1，依次递归求解就可以得到相应的结果。

（3）设 M(x)表示移动 x 个金片所需要的操作次数，则上述 N 阶汉诺塔问题可以表示成如下形式：

$$\begin{cases} M(1)=1 \\ M(N)=2M(N-1)+1 \end{cases}$$

最后可以解得 $M(N)=2^N-1$

下面给出对梵塔问题给出产生式系统描述，并讨论 N 为任意时状态空间的规模。

（1）综合数据库

定义三元组：(A, B, C)，其中 A, B, C 分别表示三根立柱，均为表，表的元素为 1～N 之间的整数，表示 N 个不同大小的盘子，数值小的数表示小盘子，数值大的数表示大盘子。表的第一个元素表示立柱最上面的柱子，其余类推。

（2）规则集

为了方便表示规则集，引入以下几个函数：

first(L)：取表的第一个元素，对于空表，first 得到一个很大的大于 N 的数值。

tail(L)：取表除了第一个元素以外，其余元素组成的表。

cons(x, L)：将 x 加入到表 L 的最前面。

规则集：

r1: IF (A, B, C) and (first(A) < first(B)) THEN (tail(A), cons(first(A), B), C)

r2: IF (A, B, C) and (first(A) < first(C)) THEN (tail(A), B, cons(first(A), C))

r3: IF (A, B, C) and (first(B) < first(C)) THEN (A, tail(B), cons(first(B), C))

r4: IF (A, B, C) and (first(B) < first(A)) THEN (cons(first(B), A), tail(B), C)

r5: IF (A, B, C) and (first(C) < first(A)) THEN (cons(first(C), A), B, tail(C))

r6: IF (A, B, C) and (first(C) < first(B)) THEN (A, cons(first(C), B), tail(C))

(3)初始状态：((1，2，...，N)，()，())

(4)结束状态：((),()，(1，2，...，N))

问题的状态规模：每一个盘子都有三种选择：在 A 上、或者在 B 上、或者在 C 上，共 N 个盘子，所以共有3^N种可能。即问题的状态规模为3^N。

2.14 解答：

(1)定义谓词 G(x,y):x 比 y 大，个体有张三（zhang)、李四（li)，将这些个体带入谓词中，得到 G(zhang,li)和¬G(zhang,li),根据语义用逻辑连接词将它们联结起来就得到表示上述知识的谓词公式：~~G(zhang,li)~~ ¬G(zhang,li)。

(2) 定义谓词 Marry(x,y):x 和 y 结婚，Male(x):x 是男的，Female(x)：x 是女的。个体有甲(A)、乙(B)，将这些个体带入谓词中，得到 Marry(A,B)、Male(A)、Female(B)以及 Male(A)、Female(B),根据语义用逻辑连接词将它们联结起来就得到表示上述知识的谓词公式：

$$Marry(A,B) \rightarrow (Male(A) \wedge Female(B)) \vee (Male(B) \wedge Female(A))$$

(3) 定义谓词 Honest(x):x 是诚实的，Lying(x):x 会说谎。个体有张三（zhang)，将这些个体带入谓词中，得到 Honest(x)、¬Lying(x)、Lying(zhang)、¬Honest(zhang),根据语义用逻辑连接词将它们联结起来就得到表示上述知识的谓词公式：

$$\forall X \quad (Honest(x) \rightarrow \neg Lying(x)) \rightarrow (Lying(zhang)$$

¬Honest(zhang))

第三章　问题求解方法

3.1 答：深度优先搜索与广度优先搜索的区别在于：在对节点 n 进行扩展时，其后继节点在 OPEN 表中的存放位置不同。广度优先搜索是将后继节点放入 OPEN 表的末端，而深度优先搜索则是将后继节点放入 OPEN 表的前端。广度优先搜索是一种完备搜索，即只要问题有解就一定能够求出，而深度优先搜索是不完备搜索。

在不要求求解速度且目标节点的层次较深的情况下，广度优先搜索优于深度优先搜索；在要求求解速度且目标节点的层次较浅的情况下，深度优先搜索优于广度优先搜索。

广度优先的正例：积木问题；深度优先的正例：邮递员问题，反例：国际象棋。

3.2 答：衡量标准为：这组子状态中有没有目标状态，如果有，则选择该节点并且搜索成功；若没有，则按照某种控制策略从已生成的状态中再选择一个状态作为当前状态重复搜索过程。

3.3 答：(1)广度优先搜索：该程序必须找到解，并且最好是最优解；
(2)广度优先搜索：医生要根据病人的各种病状判断病人的病；
(3)深度优先搜索：该程序要求一定要找到目标路径；
(4)深度优先搜索：该程序要求找到最优解；
(5)广度优先搜索：不能确定它们是否等同，既不能确定它们是否有等同解。

3.4 答：对于四皇后问题，如果放一个皇后的耗散值为 1 的话，则任何一个解的耗散值都是 4。因此如果 h 是对该耗散值的估计，是没有意义的。对于像四皇后这样的问题，启发函数应该是对找到解的可能性的评价。利用一个位置放皇后后，消去的对角线的长度来进行评价。

3.5 答：定义 ***h1=M+C-2B***，其中 M，C 分别是在河的左岸的传教士人数和野人人数。B＝1 表示船在左岸，B＝0 表示船在右岸。也可以定义 ***h2=M+C***。h1 是满足 A*条件的，而 h2 不满足。

要说明 h2＝M+C 不满足 A*条件是很容易的，只需要给出一个反例就可以了。比如状态(1, 1, 1)，h2=M+C=1+1=2，而实际上只要一次摆渡就可以达到目标状态，其最优路径的耗散值为 1。所以不满足 A*的条件。

下面我们来证明 h1＝M+C-2B 是满足 A*条件的。

我们分两种情况考虑。先考虑船在左岸的情况。如果不考虑限制条件，也就是说，船一次可以将三人从左岸运到右岸，然后再有一个人将船送回来。这样，船一个来回可以运过河 2 人，而船仍然在左岸。而最后剩下的三个人，则可以一次将他们全部从左岸运到右岸。所以，在不考虑限制条件的情况下，也至少需要摆渡 $\left\lceil \frac{M+C-3}{2} \right\rceil \times 2+1$ 次。其中分子上的"－3"表示剩下三个留待最后一次运过去。除以"2"是因为一个来回可以运过去 2 人，需要 $\frac{M+C-3}{2}$ 个来回，而"来回"数不能是小数，需要向上取整，这个用符号$\lceil\ \rceil$表示。而乘以"2"是因为一个来回相当于两次摆渡，所以要乘以 2。而最后的"＋1"，则表示将剩下的 3 个运过去，需要一次摆渡。

化简有：

$$\left\lceil \frac{M+C-3}{2} \right\rceil \times 2+1 \geq \frac{M+C-3}{2} \times 2+1 = M+C-3+1 = M+C-2$$

再考虑船在右岸的情况。同样不考虑限制条件。船在右岸，需要一个人将船运到左岸。因此对于状态(M，C，0)来说，其所需要的最少摆渡数，相当于船在左岸时状态(M+1，C，1)或(M，C+1，1)所需要的最少摆渡数，再加上第一次将船从右岸送到左岸的一次摆渡数。因此所需要的最少摆渡数为：(M+C+1)-2+1 。其中(M+C+1)的"＋1"表示送船回到左岸的那个人，而最后边的"＋1"，表示送船到左岸时的一次摆渡。化简有：(M+C+1)-2+1=M+C。

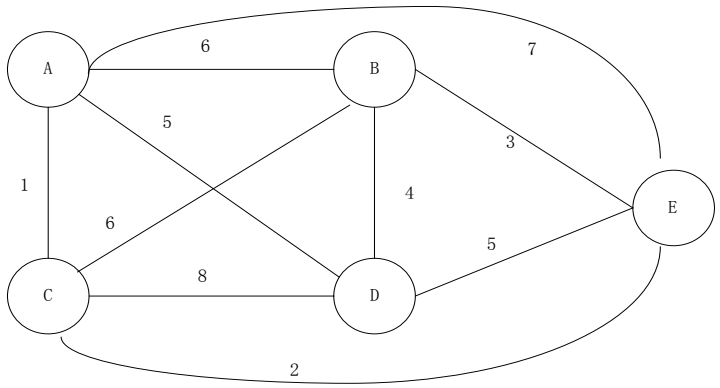
综合船在左岸和船在右岸两种情况下，所需要的最少摆渡次数用一个式子表示为：M+C-2B。其中 B＝1 表示船在左岸，B＝0 表示船在右岸。 由于该摆渡次数是在不考虑限制条件下，推出的最少所需要的摆渡次数。因此，当有限制条件时，最优的摆渡次数只能大于等于该摆渡次数。所以该启发函数 h 是满足 A*条件的。

3.6 答：在搜索期间改善 h 函数，是一种动态改变 h 函数的方法。像改进的 A*算法中，对 NEXT 中的节点按 g 值的大小选择待扩展的节点，相当于令这些节点的 h＝0，就是动态修改 h 函数的一种方法。

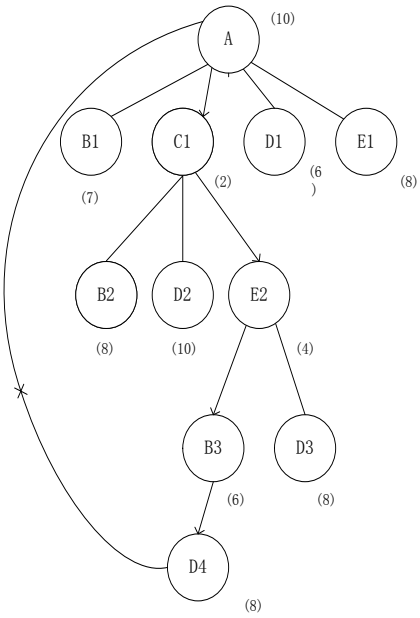
由定理 2：若 h(n)满足单调限制，则由 A*所扩展的节点序列，其 f 值是非递减的，即 f(ni)≤f(nj)），当 h 满足单调条件时，A*所扩展的节点序列，其 f 是非递减的。对于任何节点 i，j，如果 j 是 i 的子节点，则有 f(i)≤f(j)。利用该性质，我们可以提出另一种动态修改 h 函数的方法：f(j)=max(f(i), f(j))

以 f(j)作为节点 j 的 f 值。f 值的改变，隐含了 h 值的改变。当 h 不满足单调条件时，经过这样修正后的 h 具有一定的单调性质，可以减少重复节点的可能性。

3.7 答：



像这种类型的问题，由于涉及到城市距离或旅行费用，所以利用代价树广度优先搜索求解。为此，首先必须将旅行交通图转换为代价树，转换方法为：从初始节点 A 开始，把与它直接相邻的节点作为他的后继节点，对其他节点也作同样的扩展，但若一个节点以作为某节点的前驱节点，则它就不能再作为该结点的后继结点。另外，图中节点除了初始节点 A 之外，其它的节点都有可能在代价树中多次出现，为了区分它们的多次出现，分别用下标 1,2…标出。但他们却是图中的同一个节点。设估价函数 f(n)=d(n)+w(n)，其中 d(n)为状态的深度，w(n)为城市间的距离。代价树如下所示：



ACEBDA

定义h1=n*k，其中n是还未走过的城市数，k是还未走过的城市间距离的最小值。h2＝$\sum_{i=1}^{n} k_i$，其中 n 是还未走过的城市数，ki 是还未走过的城市间距离中 n 个最小的距离。 显然这两个 h 函数均满足 A*条件。

3.8 答：可定义 h 为：h＝B 右边的 W 的数目

设 j 节点是 i 节点的子节点，则根据走法不同，h(i)-h(j)的值和 C(i, j)分为如下几种情况：

（1）B 或 W 走到了相邻的一个空格位置，此时： h(i)-h(j)=0, C(i,j)=1；

（2）W 跳过了 1 或 2 个 W，此时 h(i)-h(j)=0, C(i,j)=1 或 2；

（3）W 向右跳过了一个 B（可能同时包含一个 W），此时： h(i)-h(j)=-1, C(i,j)=1 或 2；

（4）W 向右跳过了两个 B，此时： h(i)-h(j)=-2, C(i,j)=2；

（5）W 向左跳过了一个 B（可能同时包含一个 W），此时： h(i)-h(j)=1, C(i,j)=1 或 2；

（6）W 向左跳过了两个 B，此时： h(i)-h(j)=2, C(i,j)=2；

（7）B 跳过了 1 或 2 个 B，此时 h(i)-h(j)=0, C(i,j)=1 或 2；

（8）B 向右跳过了一个 W（可能同时包含一个 B），此时： h(i)-h(j)=1, C(i,j)=1 或 2；

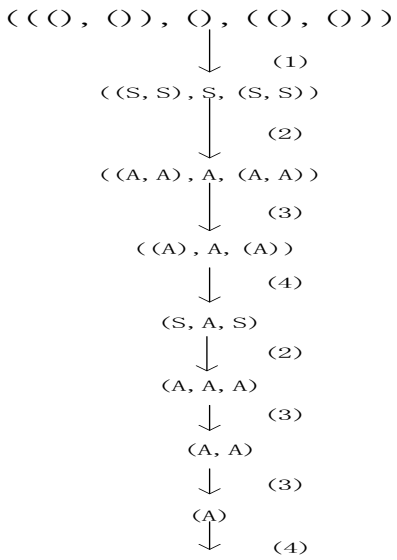
（9）B 向右跳过了两个 W，此时： h(i)-h(j)=2, C(i,j)=2；

（10）B 向左跳过了一个 W（可能同时包含一个 B），此时： h(i)-h(j)=-1, C(i,j)=1 或 2；

（11）B 向左跳过了两个 W，此时： h(i)-h(j)=-2, C(i,j)=2；

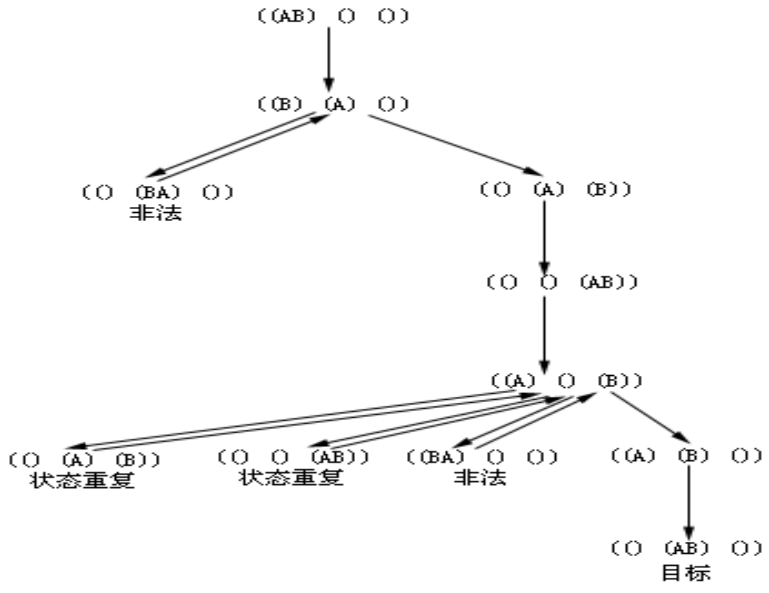
纵上所述，无论是哪一种情况，具有:h(i)-h(j)≤C(i,j)。且容易验证 h(t)=0，所以该 h 是单调的。由于 h 满足单调条件，所以也一定有 h(n)≤h*(n)，即满足 A*条件。

3.9 答：

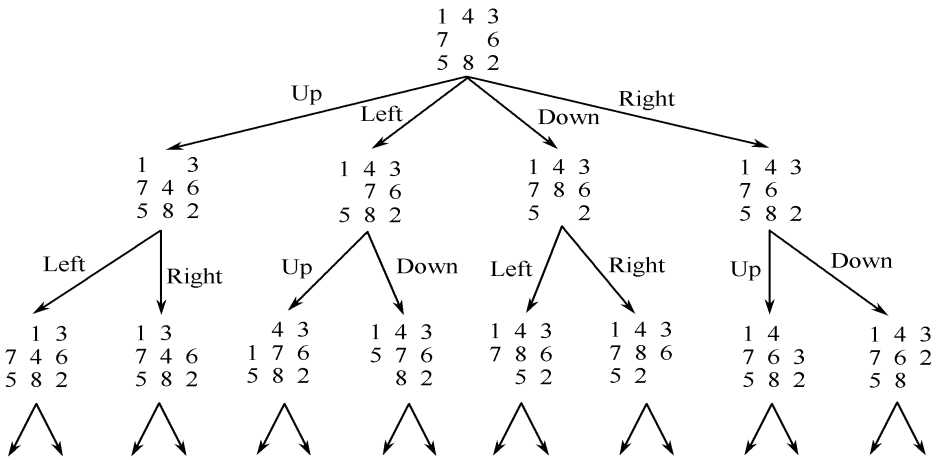


3.10 答：

***(1)*余一棋的弈法**如下：两棋手可以从 5 个钱币堆中轮流拿走一个、两个或三个钱币，拣起最后一个钱币者算输。试通过博弈证明，后走的选手必胜，并给出一个简单的特征标记来表示取胜策略。为了方便起见，用((AB)()())这样的表表示一个状态。这样得到搜索图如下：



***(2)*八数码问题** 空格：Up,Left,Down,Right



3.11 答：

(1)**与/或图的解图：**那些可解结点的子图，包含一结点到目的结点集的、连通的可解结点的子图。在问题的完整的隐含图中扩展生成出包含初始结点和目的结点集合的连通的明显子图。

(2)**算法 *AO****：必须对当前已生成出的与或图中的所有结点实施其每解点是否为可解结点的标注过程，如果起始结点被标注为可解的，则搜索过程可成功地结束；如果起始结点还不能被标注为可解的，则应当继续扩展生成结点（尽可能地记录，所有生成的结点中，哪些结点被标注了可解的，以便减少下一次标注过程的工作量）；同样地，对不可解结点也同样如此。

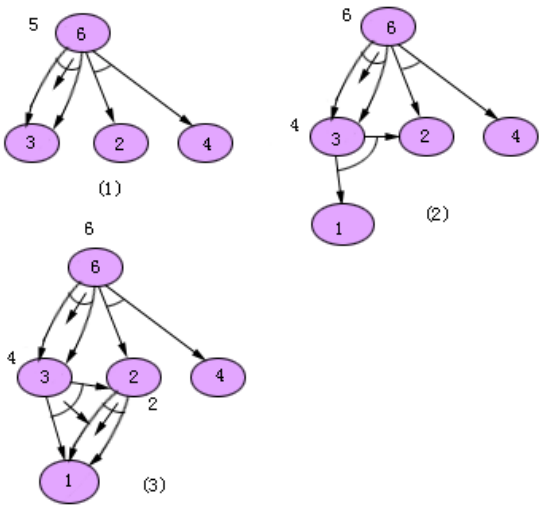
利用结点的可解/不可解性质，能从搜索图中删去可解结点的任何不可解结点的子结点；同样地，能删去不可解结点的所有的子结点（搜索这些被删除的结点是没有意义的，而只会降低搜索的效率）。两个主要过程的反复：

自上而下的图生长过程，并通过跟踪有标记的连接符寻找一个候选局部解图

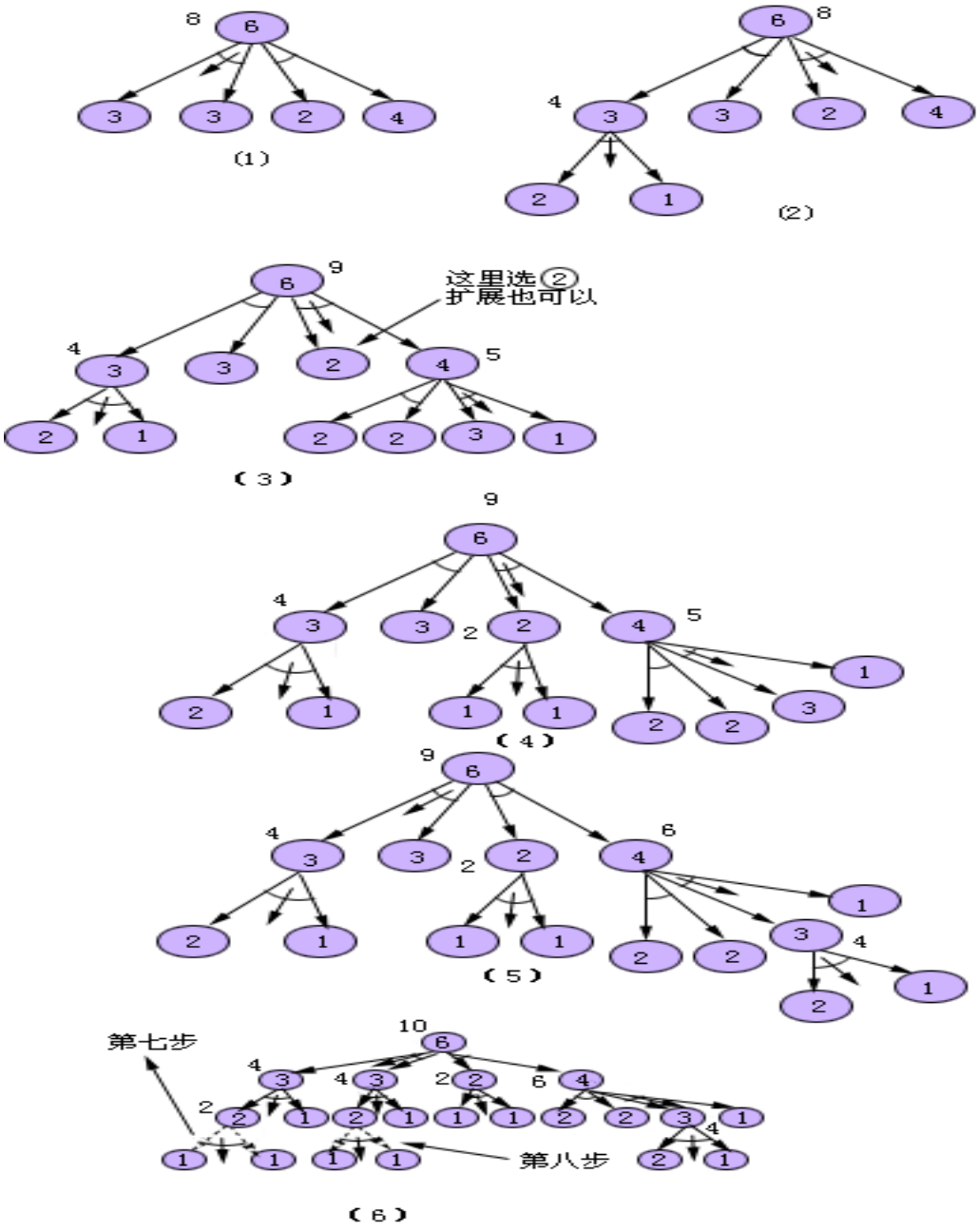
自下而上的估价函数值的修正、连接符的标记和 SOLVED 的标注过程

(3)

3.12 答：此题要求按照课中例题的方式，给出算法，以下是每个循环结束时的搜索图。



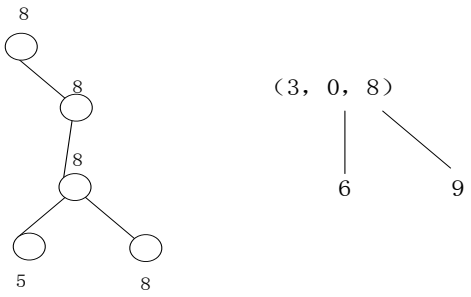
上面这种做法比较简单，也可以如下做：



3.13 答：略

3.14 答：博弈搜索通常被限制在一定的范围，搜索的目标是确定一步好的走法（好棋），等对手回手后，再继续搜索。因此，博弈搜索过程总是由当前状态向目标状态搜索，而不是由目标状态向当前状态搜索。这类博弈的实例有西洋跳棋等。

3.15 答：8—{（3，0，8）}—{（7，8，3）、（0，6）、（8，9）}—{（7,6）、（8,6,5）、（2,3）、（0,-2）、（6,2）、（5,8）、（9,2）}



3.16 答： 见上图

3.17 答：略

3.18 答：α－β 剪裁算法.

α 剪裁－若极小层的 β<=α(先辈层)则中止这个 MIN 以下的搜索.

β 剪裁－若极大层的 α>β(先辈层)则中止这个 MAX 以下的搜索

算法如下：

```
double alphabeta( int depth, double alpha, double beta, Position p);
{/*  alpha 是 MAX 的当前值  beta 是 MIN 的当前值,depth  是在搜索树中的深度,p 是所求结点的位置*/
double  t;
if( depth=0 )  return  evaluate(p); /*  如果 P 是叶结点,算出 P 的值  */
for( i=1; i <= w; i++ )
{  t = alphabeta( depth - 1, beta,alpha, pi );
if( t> alpha&&MAX)
{if(t> beta)  return  t;  /*直接返回*/
   else  alpha  =  t;}
if( t<alpha&&MIN)
{if(t< beta)  return  t;  /*直接返回*/
   else  alpha=t;}
}
return  alpha;
}
```

3.19-3.22 答：略

第四章 基本的推理技术

4.1 答：

(1)**推理**：按照某种策略从已有事实和知识推出结论的过程。

(2)**正向推理** ⟶

正向推理（事实驱动推理）是由已知事实出发向结论方向的推理。

基本思想是：系统根据用户提供的初始事实，在知识库中搜索能与之匹配的规则即当前可用的规则，构成可适用的规则集 RS，然后按某种冲突解决策略从 RS 中选择一条知识进行推理，并将推出的结论作为中间结果加入到数据库 DB 中作为下一步推理的事实，在此之后，再在知识库中选择可适用的知识进行推理，如此重复进行这一过程，直到得出最终结论或者知识库中没有可适用的知识为止。

正向推理简单、易实现，但目的性不强，效率低。需要用启发性知识解除冲突并控制中间结果的选取，其中包括必要的回溯。由于不能反推，系统的解释功能受到影响。

(3)**反向推理◄————**

反向推理是以某个假设目标作为出发点的一种推理，又称为目标驱动推理或逆向推理。

反向推理的基本思想是：首先提出一个假设目标，然后由此出发，进一步寻找支持该假设的证据，若所需的证据都能找到，则该假设成立，推理成功；若无法找到支持该假设的所有证据，则说明此假设不成立，需要另作新的假设。

与正向推理相比，反向推理的主要优点是不必使用与目标无关的知识，目的性强，同时它还有利于向用户提供解释。反向推理的缺点是在选择初始目标时具有很大的盲目性，若假设不正确，就有可能要多次提出假设，影响了系统的效率。

反向推理比较适合结论单一或直接提出结论要求证实的系统。

(4)推理方式分类

- 演绎推理、归纳推理、默认推理
- 确定性推理、不精确推理
- 单调推理、非单调推理
- 启发式推理、非启发式推理

4.2 答：

(1) 在推理过程中，系统要不断地用数据库中的事实与知识库中的规则进行匹配，当有一个以上规则的条件部分和当前数据库相匹配时，就需要有一种策略来决定首先使用哪一条规则，这就是冲突解决策略。冲突解决策略实际上就是确定规则的启用顺序。

(2) 冲突解决策略：专一性排序、规则排序、数据排序、就近排序、上下文限制、按匹配度排序、按条件个数排序

4.3 答：归结反演就是利用归结和反演实现定理的证明。具体过程如下：

(1) 将定理证明的前提谓词公式转化为子句集 F。

(2) 将求证的目标表示成合适的谓词公式 G（目标公式）。

(3) 将目标公式的否定式 ¬G 转化成子句的形式，并加入到子句集 F 中，得到子句集 S。

(4) 应用归结原理对子句集 S 中的子句进行归结，并把每次归结得到的归结式都并入 S 中。如此反复进行，若归结得到一个空子句 NIL，则停止归结，证明了 G 为真。

4.4 答：略

4.5 答：

（1）$(\forall x)(\forall y)[P(x,y)\rightarrow Q(x,y)]=(\forall x)(\forall y)[\sim P(x,y)\vee Q(x,y)]=\{\sim P(x,y)\vee Q(x,y)\}$

子句集为$\sim P(x,y)\vee Q(x,y)$

（2）$(\forall x)(\exists y)[P(x,y)\vee Q(x,y)\rightarrow R(x,y)]=(\forall x)(\exists y)[\sim P(x,y)\wedge\sim Q(x,y)\vee R(x,y)]=\{\sim P(x)\vee P(x)\}$

=(∀ x)[~P(x,f(x)) ∧ ~Q(x, f(x)) ∨ R(x, f(x))]= ~P(x,f(x)) ∧ ~Q(x, f(x)) ∨ R(x, f(x))= [~P(x,f(x))∨R(x, f(x))]∧[~Q(x, f(x))∨R(x, f(x))]= [~P(x,f(x))∨R(x, f(x))]∧[~Q(y, f(y))∨R(y, f(y))]

子句集为~P(x,f(x))∨R(x, f(x))和~Q(y, f(y))∨R(y, f(y))

（3）(∀x){(∀y)P(x,y)→~ (∀y)[Q(x,y) →R(x,y)]}=(∀x)){(∃y) ~P(x,y)∨~ (∀y)[Q(x,y) →R(x,y)]}= (∀x) [(∃y) ~P(x,y)∨(∃y)[~Q(x,y)∨R(x,y)]= (∀x) [~P(x,f(x))∨[~Q(x, f(x))∨R(x, f(x))]= ~P(x,f(x))∨~Q(x, f(x))∨R(x, f(x))

子句集为~P(x,f(x))∨~Q(x, f(x))∨R(x, f(x))

4.6 答：

(1)

(2){A/x, A/y, A/z, A/w, A/u}

(3)

4.7 答：

（1）（∃x）{[P（x）→P（A）]∧[P（x）→P（B）]}

目标取反化子句集：

~（∃x）{[P（x）→P（A）]∧[P（x）→P（B）]}

~（∃x）{[~P（x）∨P（A）]∧[~P（x）∨P（B）]}

（∀x）{[P（x）∧~P（A）]∨[P（x）∧~P（B）]}

（∀x）{[P（x）∧~P（A）]∨P（x）}∧{[P（x）∧~P（A）]∨~P（B）}}

（∀x）{P（x）∧[~P（A）∨P（x）]∧[P（x）∨~P（B）]∧[~P（A）∨~P（B）]}

P（x）∧[~P（A）∨P（x）]∧[P（x）∨~P（B）]∧[~P（A）∨~P（B）]

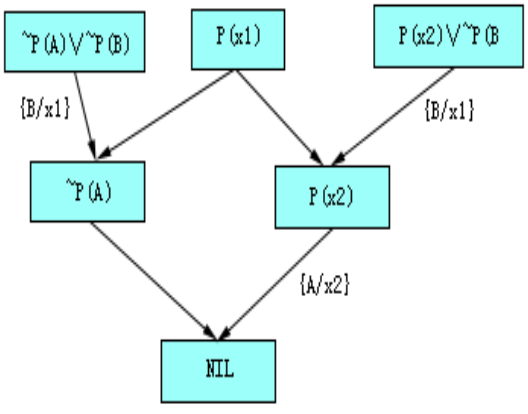
得子句集：

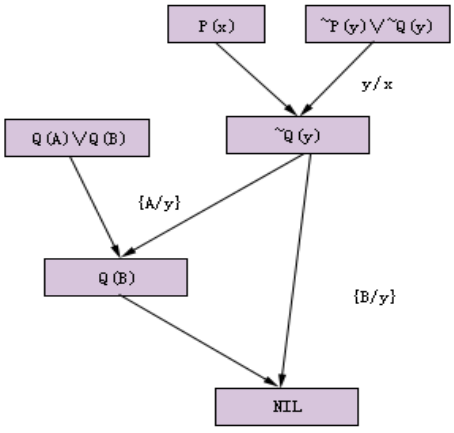
1, P(x1)

2, ~P(A)∨P{x2}

3, P(x3)∨~P(B)

4, ~P(A)∨~P(B)





（2）（∀x）{P（x）∧[Q（A）∨Q（B）]}→（ϖx）[P（x）∧Q（x）]

目标取反化子句集：

~{(∀x){P(x)∧[Q(A)∨Q(B)]}→(ϖx)[P(x)∧Q(x)]}

~{~{(∀x)P(x)∧[Q(A)∨Q(B)]}∨(ϖx)[P(x)∧Q(x)]}

{(∀x)P(x)∧[Q(A)∨Q(B)]}∧(∀x)[~P(x)∨~Q(x)]}

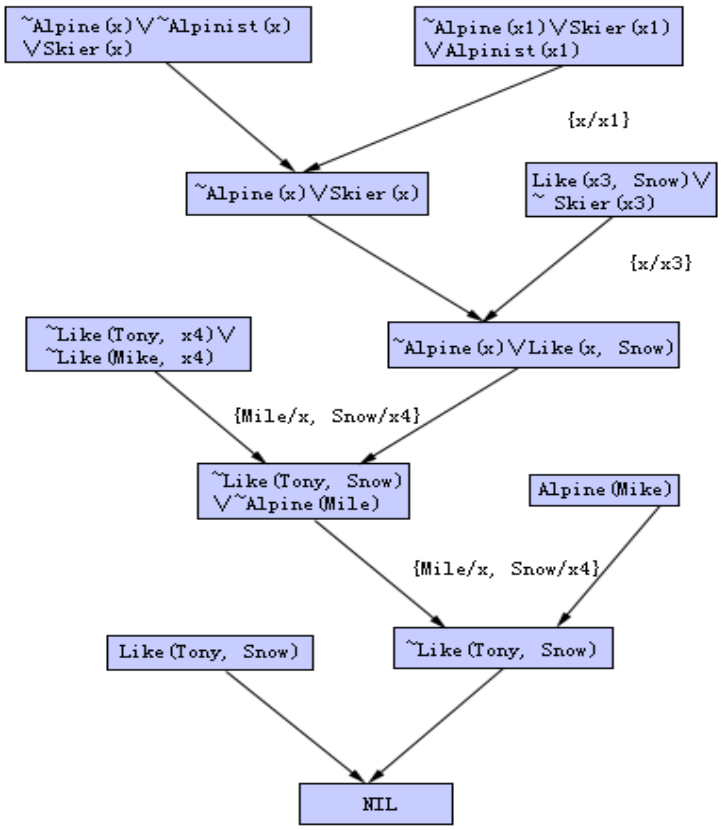
{(∀x)P(x)∧[Q(A)∨Q(B)]}∧(∀y)[~P(y)∨~Q(y)]}
(∀x)(∀y){P(x)∧[Q(A)∨Q(B)]∧[~P(y)∨~Q(y)]}
P(x)∧[Q(A)∨Q(B)]∧[~P(y)∨~Q(y)]
得子句集：
1，P(x)
2，Q(A)∨Q(B)
3，~P(y)∨~Q(y)

4.8 答：

4.9 答：
答：我们用 Skier(x)表示 x 是滑雪运动员，Alpinist(x)表示 x 是登山运动员，Alpine(x)表示 x 是 Alpine 俱乐部的成员。
问题用谓词公式表示如下：
已知：
(1) Alpine(Tony)
(2) Alpine(Mike)
(3) Alpine(John)
(4) (∀x){Alpine(x)→[Skier(x)∨Alpinist(x)]}
(5) (∀x){Alpinist(x)→~Like(x, Rain)}
(6) (∀x){~Like(x, Snow)→~ Skier(x)}
(7) (∀x){Like(Tony, x)→~Like(Mike, x)}
(8) (∀x){~Like(Tony, x)→Like(Mike, x)}
(9) Like(Tony, Snow)
(10) Like(Tony, Rain)
目标：(∃x){Alpine(x)∧Alpinist(x)∧~Skier(x)}

化子句集：
(1) Alpine(Tony)
(2) Alpine(Mike)
(3) Alpine(John)
(4)(∀x){Alpine(x)→[Skier(x)∨Alpinist(x)]} = (∀x){~Alpine(x)∨[Skier(x)∨Alpinist(x)]} =>~Alpine(x)∨Skier(x)∨Alpinist(x)
(5) (∀x){Alpinist(x)→~Like(x, Rain)} = (∀x){~Alpinist(x)∨~Like(x, Rain)} =>~Alpinist(x)∨~Like(x, Rain)
(6) (∀x){~Like(x, Snow)→~ Skier(x)} = (∀x){Like(x, Snow)∨~ Skier(x)} => Like(x, Snow)∨~ Skier(x)
(7)(∀x){Like(Tony, x)→~Like(Mike, x)} = (∀x){~Like(Tony, x)∨~Like(Mike, x)} =>~Like(Tony, x)∨~Like(Mike, x)
(8) (∀x){~Like(Tony, x)→Like(Mike, x)} = (∀x){Like(Tony, x)∨Like(Mike, x)} => Like(Tony, x)∨Like(Mike, x)
(9) Like(Tony, Snow) (10) Like(Tony, Rain)
目标取反：
~(∃x){Alpine(x)∧Alpinist(x)∧~Skier(x)}

= (∀x){~Alpine(x)∨~Alpinist(x)∨Skier(x)}
=>~Alpine(x)∨~Alpinist(x)∨Skier(x)
经变量换名后，得到子句集：
{Alpine(Tony), Alpine(Mike), Alpine(John), ~Alpine(x1)∨Skier(x1)∨Alpinist(x1), ~Alpinist(x2)∨~Like(x2, Rain), Like(x3, Snow)∨~ Skier(x3), ~Like(Tony, x4)∨~Like(Mike, x4), Like(Tony, x5)∨Like(Mike, x5), Like(Tony, Snow), Like(Tony, Rain), ~Alpine(x)∨~Alpinist(x)∨Skier(x)}
归结树如下：



4.10 答：基于规则的演绎推理可分为正向演绎推理、反向演绎推理和正反向混合演绎推理。

在**正向演绎推理**中，作为 F 规则用的蕴含式对事实的总数据库进行操作运算，直至得到该目标公式的一个终止条件为止。事实——→目标公式

在**反向演绎推理**中，作为 B 规则用的蕴含式对目标的总数据库进行操作运算，直至得到包含这些事实的终止条件为止。目标公式 ——→事实

4.11 答：

第五章 不精确推理

5.1 答：不精确推理是建立在非经典逻辑基础上的一种推理，是基于不确定性知识的推理。不精确推理就是从不确定性的初始事实（证据）出发，通过运用不确定性的知识，最终推出具有一定程度的不确定性却是合理或者近乎合理的结论的思维过程。

在不精确推理中，知识和证据都具有不确定性，这为推理机的设计与实现增加了复杂度和难度。它除了必须解决推理方向、推理方法和控制策略等基本问题外，一般还需要解决不确定性的表示、不确定性的匹配和不确定性的更新算法等问题。

5.2 答：有明确定义但不一定出现的事件中包含的不确定性称为随机性，他不因人的主观意思变化，由事物本身的因果律决定。不精确推理就是表示和处理随机性的推理方法。

5.3 答：

(1)当有一个证据 E1 时，根据 Bayes 公式，可得

$$P(H_1 \mid E_1) = \frac{P(H_1)\cdot P(E_1 \mid H_1)}{P(H_1)\times P(E_1 \mid H_1)\cdot + P(H_2)P(E_1 \mid H_2) + P(H_3)P(E_1 \mid H_3)}$$

=0.4*0.5/（0.4*0.5+0.3*0.3+0.3*0.5）=0.2/0.44=0.45

同理可得：$P(H_1 \mid E_2)$=0.09/0.44=0.20　　$P(H_1 \mid E_3)$=0.15/0.44=0.34

这说明，由于证据 E1 的出现，H1 和 H3 成立的可能性有所增加，而 H2 成立的可能性有所下降。

(2)当证据 E1、E2 同时出现时，根据多证据情况下的 Bayes 公式，可得

$$P(H_1 \mid E_1E_2) = \frac{P(E_1 \mid H_1)\cdot P(E_2 \mid H_1)\cdot P(H_1)}{P(E_1 \mid H_1)\cdot P(E_2 \mid H_1)\cdot P(H_1) + P(E_1 \mid H_2)\cdot P(E_2 \mid H_2)\cdot P(H_2) + P(E_1 \mid H_3)\cdot P(E_2 \mid H_3)\cdot P(H_3)}$$

=0.14/（0.14+0.162+0.009）=0.59

同理可得：$P(H_2 \mid E_1E_2) = 0.34$　$P(H_3 \mid E_1E_2) = 0.064$

这说明，由于证据 E1 和 E2 的出现，H1 和 H2 成立的可能性有不同程度的增加，而 H3 成立的可能性则有了较大幅度的下降。

5.4 答：

① LS

LS 为规则的充分性量度，它反映 E 的出现对 H 的支持程度。当 LS=1 时，O (H|E)= O(H)，说明 E 对 H 没有影响；当 LS>1 时，O(H|E)>O(H)，说明 E 支持 H，且 LS 越大，E 对 H 的支持越充分，若 LS 为∞，则 E 为真时 H 就为真；当 LS<1 时，O(H|E)<O(H)，说明 E 排斥 H，若 LS 为 0，则 O(H|E)=0，即 E 为真时 H 就为假

② LN

LN 为规则的必要性量度，它反映 ¬E 对 H 的支持程度，即 E 的出现对 H 的必要性。当 LN=1 时，O(H|¬E)=O(H)，说明 ¬E 对 H 没有影响；当 LN>1 时，O(H|¬E)>O(H)，说明 ¬E 支持 H，且 LN 越大，¬E 对 H 的支持越充分，若 LN 为∞，则 ¬E 为真时 H 就为真；当 LN<1 时，O(H|¬E)<O(H)，说明 ¬E 排斥 H，若 LS 为 0，则 O(H|¬E)=0，即 ¬E 为真时 H 就为假

③ LS 和 LN 的关系

由于 E 和 ¬E 不会同时支持或排斥 H，所以只有以下三种情况存在：

情形 1：LS>1 且 LN<1

情形 2：LS<1 且 LN>1

情形 3：LS=LN=1

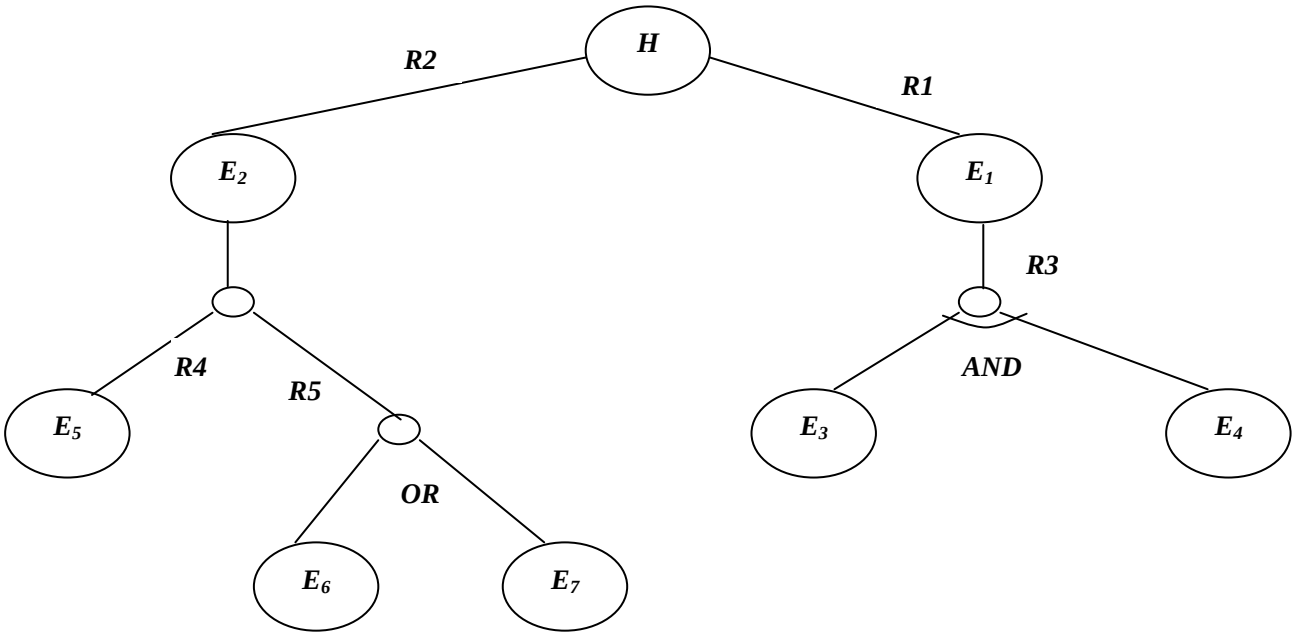
5.5 答：

5.6 答：根据经验对一个事物或现象为真的相信程度称为可信度。

规则的一般形式为：IF E THEN H （CF(H, E)）。其中，CF(H,E)是该规则的可信度，称为可信度因子或规则强度。CF(H, E)在[-1，1]上取值，它表示在已知证据 E 的情况下对假设 H 为真的支持程度。

CF(H, E)定义如下：CF(H,E)= MB(H,E)－MD(H,E)。其中，***MB***（Measure Belief）称为信任增长度，表示因证据 E 的出现而增加对假设 H 为真的信任增加程度[MB(H, E)>0 P(H|E)>P(H)]；***MD***（Measure Disbelief）称为不信任增长度，表示因证据 E 的出现对假设 H 为假的信任减少的程度[MD(H, E)>0 P(H|E)<P(H)]。

5.7 答：



（1）求证据 E3、E4 逻辑组合的可信度

CF(E3 AND E4)=min{CF(E3),CF(E4)}=min{0.5,09}=0.8

(2）根据规则 R3 求 CF(E1)

CF(E1)=0.9×max{0, CF(E3 AND E4)}=0.9×0.8=0.72

（3）求证据 E6、E7 逻辑组合的可信度

CF(E6 OR E7)=max{CF(E6),CF(E7)}=max{0.1,0.5}=0.5

（4）根据规则 R5 求 CF1(E2)

CF1(E2)=-0.3×max{0, CF(E6 OR E7)}=-0.3×0. 5=-0.15

（5）根据规则 R4 求 CF2(E2)

CF2(E2)=0.7×max{0, CF(E5)}=0.7×0. 8=-0.56

（6）根据规则 R5 求 CF1(E2)

CF1(E2)=-0.3×max{0, CF(E6 OR E7)}=-0.3×0. 5=-0.15

（7）组合由独立证据导出的假设 E2 的可信度 CF1(E2)、CF2(E2)，得到 E2 的综合可信度 CF(E2)

CF(E2)=CF1(H)+CF2(H)=0.56-0.15=0.41

（8）根据规则 R1 求 CF1(H1)

CF1(H1)=0.8×max{0, CF(E1)}=0.8×0. 72=-0.576

（8）根据规则 R2 求 CF2(H1)

CF2(H1)=0.9×max{0, CF(E2)}=0.9×0. 41=-0.369

（9）组合由独立证据导出的假设 H1 的可信度 CF1(H1)、CF2(H1)，得到 H1 的综合可信度 CF(H1)

CF(H)=CF1(H)+CF2(H)-CF1(H) CF2(H)=0.576+0.369-0.576×0.369=0.723

5.8 答：已经出现但难以给出精确定义的事件中包含的不确定性称为模糊性，是由事物的概念界限模糊和人的主观推理与判断产生的。而有明确定义但不一定出现的事件中包含的不确定性称为随机性，他不因人的主观意思变化，由事物本身的因果律决定。不精确推理就是表示和处理随机性的推理方法。两者之间有着本质的区别。

5.9 答：$A \cap B$ =0.5/x1+0.65/x2+0.8/x3+0.9/x4+0.7/x5；

$A \cup B$ =0.85/x1+0.7/x2+0.9/x3+0.98/x4+0.77/x5；

$\neg A$ =0.15/x1+0.3/x2+0.1/x3+0.1/x4+0.3/x5。

5.10 答：

$$A \circ B = \begin{bmatrix} 0.7 & 0.4 \\ 0.7 & 0.4 \\ 0.5 & 0.4 \end{bmatrix}$$

5.11 答：模糊推理实质上是在模糊集合上进行操作。在同一论域中，证据的“与”、“或”、“非”运算通常可以对应于模糊集的交、并、求补操作；不同论域中的逻辑运算一般需拓广至笛卡尔意义下的相应操作。逻辑推理是通过逻辑蕴含实现的。

设 U 和 V 为两个论域，A 是 U 上的模糊子集，B 是 V 上的模糊子集，则规则

IF A THEN B

可以定义为 U×V 上的一个模糊关系：$(A \times B) \bigcup (\neg A \times V)$

或等价表示成：$A \to B = \int_{U \times V} \max(\min(\mu_A(x), \mu_B(y)), 1 - \mu_A(x)) / (x, y)$

第六章 *PROLOG* 语言

6.3 答：（1）目标不成功

（2）目标不成功

（3）目标成功，x,y,z 被例化为 x1,y1,z1

（4）目标不成功

（5）目标不成功

6.4 答：poglog 规则

```
Is_mother(x):_mother(x,y)
Is_father(x):_father(x,y)
Is_son(x):_father(y,x),male(x)
Grandpa(x,y):_father(x,y1),father(y1,y)
Sibling(x,y):_diff(x,y),mother(z1,x),mother(z1,y),father(z2,x), father(z2,y),parent(x,y)
```

6.5 答：(1)family1.cl
文件头:

```
/*****************************************************************************
                        Copyright (c) Sabu Francis Associates
*****************************************************************************/
class family1
    open core
    predicates
        classInfo : core::classInfo.
        % @short Class information    predicate.
        % @detail This predicate represents information predicate of this class.
        % @end
    predicates
        run : core::runnable.
end class family1
```

(2)family1.pack
文件头:

```
/*****************************************************************************
                        Copyright (c) Sabu Francis Associates
*****************************************************************************/
#include @"family1.ph"
% privately used packages
#include @"pfc\filesystem\filesystem.ph"
#include @"pfc\application\exe\exe.ph"
#include @"pfc\console\console.ph"
#include @"pfc\exception\exception.ph"
% private interfaces
% private classes
% implementations
#include @"family1.pro"
```

(3)family1.ph
文件头:

```
/*****************************************************************************
                        Copyright (c) Sabu Francis Associates
*****************************************************************************/
#requires @"family1.pack"
% publicly used packages
#include @"pfc\core.ph"
% exported interfaces
% exported classes
#include @"family1.cl"
```

(4) family1.prj6
文件头:

```
/*****************************************************************************
                        Copyright (c) Sabu Francis Associates
*****************************************************************************/
implement family1
    open core
    constants
        className = "family1".
        classVersion = "$JustDate: $$Revision: $".
    clauses
        classInfo(className, classVersion).
    domains
        gender = female(); male().
    class facts - familyDB
        person : (string Name, gender Gender).
        parent : (string Person, string Parent).
    class predicates
        father : (string Person, string Father) nondeterm anyflow.
    clauses
        father(Person, Father) :-
            parent(Person, Father),
            person(Father, male()).
    class predicates
        grandFather : (string Person, string GrandFather) nondeterm (o,o).
    clauses
        grandFather(Person, GrandFather) :-
            parent(Persoicates)
文件尾:
     reconsult : (string FileName).
     clauses
        reconsult(FileName) :-
            retractFactDB(familyDB),
            file::consult(FileName, familyDB).
    clauses
        run():-
            console::init(),
            stdIO::write("Load data\n"),
            reconsult("..\\fa.txt"),
            stdIO::write("\nfather test\n"),
            father(X, Y),
                stdIO::writef("% is the father of %\n", Y, X),
            fail.
        run():-
            stdIO::write("\ngrandFather test\n"),
            grandFather(X, Y),
```

```
            stdIO::writef("% is the grandfather of %\n", Y, X),
        fail.
    run():-
        stdIO::write("\nancestor of Pam test\n"),
        X = "Pam",
        ancestor(X, Y),
            stdIO::writef("% is the ancestor of %\n", Y, X),
        fail.
    run():-
        stdIO::write("End of test\n").
end implement family1
goal
    mainExe::run(family1::run).
```

(5)family1.pro

补充习题：

1.编写一 Prolog 程序使得你能和计算机“交谈”（Conversations with a computer）。下图显示了对话时的情景，字体加粗的语句表示用户从键盘输入的内容，其他则是计算机的回答。

HELLO

HI

DO YOU WANT TO TALK

NO I WANT TO SLEEP

YOU ARE A STUPID COMPUTER

I AM A NICE COMPUTER

```
1.答：domains                       /*领域段，说明程序要用到的数据类型*/
   words=string
   sentence=words*
predicates                      /*谓词段，说明程序要用到的谓词名和参数*/
   nondeterm   talk
   nondeterm   human(sentence)
   nondeterm   answer(sentence)
   nondeterm   tolist(words,sentence)
nondeterm   process(words)
nondeterm   change(words,words)
clauses                         /*子句段，说明程序要用到的事实和规则*/
talk:-human(X),answer(X),talk.
   human(Y):-readln(X),tolist(X,Y).
tolist(Str,[H|T]):-fronttoken(Str,H,Str1),!,tolist(Str1,T).
   tolist(_,[]).
   answer([]):-nl,nl.
answer([Y|Z]):-process(Y),answer(Z).
process(X):-change(X,Y),write(Y),write(" ").
change("HELLO","HI").
```

```
change("DO","NO").
change("YOU","I").
change("TALK","SLEEP").
change("ARE","AM").
change("STUPID","NICE").
change(X,X).
goal                          /*目标段，说明程序要求解的目标*/
talk.
```

2.有一个农夫带着一只狼、一只山羊和一筐白菜过河，要从南岸过河到北岸。岸边有一条小船，农夫可以独自划着小船过河，或者每次带其中的一样东西过河。问题是如果他留下狼和羊在一起，那么狼就会把羊吃掉；如果留下羊和白菜，那么羊就会把白菜吃掉。农夫如何才能安全地将全部东西运到河对岸？

2.答：**设计方案**

可以用 farmer、wolf、goat 和 cabbage 分别表示问题中的农夫、狼、山羊和白菜等对象。状态描述了各个成员的位置，可以用字母 n 和 s 分别表示北岸和南岸，例如，可以用 location(s, n, n, s)表示农夫、狼、山羊和白菜分别在南岸、北岸、北岸和南岸。动作描述了农夫的摆渡操作，根据题意，一共只有四种动作，分别是农夫独自划船、农夫带着狼划船、农夫带着山羊划船和农夫带着白菜划船。

问题求解的目标是一个特定的动作序列（即渡河程序），而每个动作都将当前状态转变成另一个新的状态，所以状态的变化过程。可以反应出动作执行的过程。因此，可以将求解动作序列的问题转换成求解状态变化过程的问题，状态变化过程求出来后，只要用某种方法将状态变化转换成相应的动作序列输出即可。我们用一个表 X 表示这个未知的状态变化过程，X 的条件是它必须由和平的状态（即不会出现一个对象将另一个对象吃掉的情况）组成。开始时所有对象都在南岸，到最后所有对象都在北岸。用 Prolog 规则表示就是 legal_states(location(s, s, s, s), location(n, n, n, n), States, X)，其中 States 是用来记录状态变化过程的动态表，在问题的开始，即农夫还没做任何事时，States 存储的应该是初始状态；到最后，即农夫摆渡结束时，States 中所存储的就是目标动作序列所对应的状态变化过程。

实施方案

```
domains /*领域段，说明程序要用到的数据类型*/
bt=boat(symbol,symbol)
ln=location(symbol,symbol,symbol,symbol)
lists=ln*
predicates /*谓词段，说明程序要用到的谓词名和参数*/
opposite(symbol,symbol)
safe_goat(symbol,symbol,symbol,symbol)
safe_cabbage(symbol,symbol,symbol,symbol)
is_peaceful(symbol,symbol,symbol,symbol)
transform(ln,ln)
legal_states(ln,ln,lists,lists)
member(ln,lists)
write_state(ln,ln)
write_actions(lists)
```

```
boat(ln,ln)
clauses /*子句段，说明程序要用到的事实和规则*/
opposite(s,n).
opposite(n,s).
safe_goat(X,_,X,_).
safe_goat(F,X,Y,_):-opposite(X,Y),F<>Y.
safe_cabbage(X,_,_,X).
safe_cabbage(F,_,X,Y):-opposite(X,Y),F<>Y.
is_peaceful(F,W,G,C):-safe_goat(F,W,G,C),safe_cabbage(F,W,G,C).
member(X,[X|_]).
member(X,[_|T]):-member(X,T).
transform(location(X,W,G,C),location(Y,W,G,C)):-opposite(X,Y).
transform(location(X,X,G,C),location(Y,Y,G,C)):-opposite(X,Y).
transform(location(X,W,X,C),location(Y,W,Y,C)):-opposite(X,Y).
transform(location(X,W,G,X),location(Y,W,G,Y)):-opposite(X,Y).
legal_states(location(F0,W0,G0,C0),location(F,W,G,C),States,X):-
transform(location(F0,W0,G0,C0),location(F1,W1,G1,C1)),
is_peaceful(F1,W1,G1,C1),not(member(location(F1,W1,G1,C1),States)),
legal_states(location(F1,W1,G1,C1),location(F,W,G,C),[location(F1,W1,G1,C1)|States],X).
legal_states(location(F,W,G,C),location(F,W,G,C),L,L).
write_state(location(X,W,G,C),location(Y,W,G,C)):-!,write("The farmer crosses the river himself"),nl.
write_state(location(X,X,G,C),location(Y,Y,G,C)):-!,write("The farmer crosses the river with the wolf"),nl.
write_state(location(X,W,X,C),location(Y,W,Y,C)):-!,write("The farmer crosses the river with the goat"),nl.
write_state(location(X,W,G,X),location(Y,W,G,Y)):-!,write("The farmer crosses the river with the cabbage"),nl.
write_actions([]).
write_actions([H1,H2|T]):-write_state(H1,H2),write_actions([H2|T]).
boat(location(F0,W0,G0,C0),location(F,W,G,C)):-
legal_states(location(F0,W0,G0,C0),location(F,W,G,C),[location(F0,W0,G0,C0)],X),
write_actions(X).
```

3.某机器人需要在一间大商店里巡逻，由于商品每天都在流动，商店的平面图经常变化。一个典型的平面路径如图所示，在这个图中，点代表机器人的工作站，线表示点与点之间适于巡逻的通道。在机器人的内部数据库中，将通过一组事实表示图的路径，例如：joined_to(a, b);joined_to(b, c);joined_to(b, d);joined_to(d, c);joined_to(c, e) 。机器人必须能够判断连接两个点的路线，例如，它应该识别出 这个序列代表从 e 到 b 的一条路线。请写出一个程序，用来证明或者反证某一个给定序列是两个给定点之间的路线。

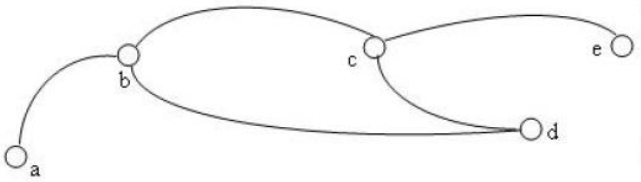


图 2 商店路线图

3.答：

理解问题

已知一个序列和一对点，要求证明（或反证）这个序列是这两个点之间一条可能的路线。可以看出，这是一个证明型问题：即证明给出的序列是给定的两点之间的一条路线。

那么什么是路线？结合图 2，我们来看几个有用的例子。

（1）b d c 这个序列是 a 和 c 两点之间的路线吗？显然不是，因为它的起始点不对。

（2）d c e 这个序列是 d 和 b 两点之间的路线吗？当然不是：它的终止点不对。

（3）a b e 是 a 和 e 之间的路线吗？不是，因为在平面图上，b 和 e 两个点之间不是直接相连的。

可见，对于一个给定的点序列，要使它成为某对特定的点之间的路线，需要满足以下三个条件：

• 序列应该从点对的第一点开始。

• 序列应该以点对的第二点结束。

• 序列应该是连通的（连通序列）——也就是说，序列中任意两个连续的点在平面图中应该是相连的。

对于给定的一个序列和一对端点，如果这三个条件都满足，那么可以确信这个序列是这两个点之间的路线。这三个条件是证明该假设的充分条件，同时也是必要条件。

设计方案

在 Prolog 中可以用一个表来表示一个点序列，例如，序列 可以表示为[e, c, d, b]；一对端点也可以用一个表表示。因此，route_between([e, c, d, b], [e, b])可以表示 egcgdgb 是 e、b 两点之间的路线。将是我们所关心的关系的一个实例。

目标是什么？我们可以写成 route_between(X, [Y, Z])，这里序列 X 和一对端点[Y, Z]是给定的，我们关心的是目标能否取得成功。

为了给 route_between 写出一条规则，我们只需要将上面写下的三个条件转换成 Prolog 就可以了，如下所示：

```
route_between(X, [Y, Z]):- begins_with(X, Y),
ends_with(X, Z),
is_connected(X).
```

如果能够写出 begins_with、ends_with 和 is_connected 这三个谓词的检验定义，我们应该能够输入询问，如 route_between([a, d, e,c, b], [a, b])，并得到一个是还是否的回答。

执行方案

(1) begins_with

该关系的一个实例就是 begins_with([b, a, d], b)。显然，这个关系要成立，端点与表头必须是完全相同的。用规则表示就是 begins_with([X|Y], X)。可以用一些询问进行试验，例如 begins_with([b, a, d,], b)等。

(2) ends_with

ends_with([b, a, d], d)就是该关系的一个实例。与 begins_with 相比，这个关系就不是那

么容易定义了，因为表的最后一个元素看起来并不像第一个元素那样特别。不过，你可以想出一个表，使得它与表[b, a, d]相关，又是从它的最后一个元素开始的吗？将原表

倒置一下，变成[d, a, b]如何？所以我们可以这样描述 ends_with：

表 X 以点 Y 结束，如果 X 的逆序从点 Y 开始。

表的倒置，即求一个表的逆序表是关于表的一个最常见的问题，这里我们不做具体讨论，其完整程序如图 4 所示。现在，我们就

可以用 reverse 关系和刚才已经定义好的 begins_with 关系，将上面的描述用 Prolog 表示成：

```
ends_with(X, Y):- reverse(X, Z),
begins_with(Z, Y).
```

可以用一些合适的询问来试验一下。

```
domains
s_list=symbol*
predicates
append(s_list,s_list,s_list)
reverse(s_list,s_list)
clauses
append([],L,L).
append([H|T],L2,[H|Tn]):-append(T,L2,Tn).
reverse([],[]).
reverse([H|T],L):-reverse(T,L1),append(L1,[H],L).
```

(3) is_connected

给这个关系举出几个例子并不困难，例如，通过图 3 我们可以看出，表[c, d, b, a]是一个连通序列，而[b, d, c, e, a]不是。用一般序列的表示方法，即[X|Y]来表示可以吗？为了使得[X|Y]是一个连通序列，对于 X 有什么要求呢？显然，X 必须与序列中下一个出现的点在平面图中是直接相连的；因此要把这个“下一个”点展开。所以我们应该用[X1,X2|Y]来表示一般序列，而不是[X|Y]；如果 X1 与 X2 相连，并且[X2|Y]是一个连通序列，那么这就是一个连通的序列。（注意这里的第二个条件，为什么只问 Y 是否连通是不够的呢？）因此，我们写出如下规则

```
is_connected([X1,X2|Y]):-linked_to(X1, X2),
is_connected([X2|Y]).
```

这里我们已经特意用 linked_to 替代 joined_to 了，为什么呢？举例来说，前面给出的 joined_to 事实声明 a 是连到 b 的，但是没有事实

对应于 b 连到 a。所以需要使用关系 linked_to，以使 Prolog 能够做出明显的推论。我们用下面两条规则来定义：

```
linked_to(X, Y):-joined_to(X, Y).
linked_to(X, Y):-joined_to(Y, X).
```

然而，我们还没有完成 is_connected 的定义。上面的规则是递归的，所以我们需要一个特殊的、非递归的情况。[X1,X2|Y]这个模式是包含两个或两个以上成员的表，一个只有一个点的表当然是“连通的”，所以我们加上 is_connected([X])，这个语句提供了一个“直接的答案”。

现在程序已经完成了，接下来就是检验我们的计算机现在确实能够解决机器人巡逻问题了，用一些询问，如 route_between([d, b,c, e], [d, e])试验一下。机器人巡逻问题的完整程序如图 5 所示，该程序在 Turbo Prolog 2.0 环境下运行通过。

```
domains /*领域段，说明程序要用到的数据类型*/
s_list=symbol*
predicates /*谓词段，说明程序要用到的谓词名和参数*/
append(s_list,s_list,s_list)
reverse(s_list,s_list)
joined_to(symbol,symbol)
linked_to(symbol,symbol)
begins_with(s_list,symbol)
ends_with(s_list,symbol)
is_connected(s_list)
route_between(s_list,s_list)
clauses /*子句段，说明程序要用到的事实和规则*/
append([],L,L).
append([H|T],L2,[H|Tn]):-append(T,L2,Tn).
reverse([],[]).
reverse([H|T],L):-reverse(T,L1),append(L1,[H],L).
joined_to(a,b).
joined_to(b,a).
joined_to(b,c).
joined_to(c,b).
joined_to(b,d).
joined_to(d,b).
joined_to(d,c).
joined_to(c,d).
joined_to(c,e).
joined_to(e,c).
linked_to(X,Y):-joined_to(X,Y).
linked_to(X,Y):-joined_to(Y,X).
begins_with([X|_],X).
ends_with(X, Y):-reverse(X,Z),begins_with(Z,Y).
is_connected([_]).
is_connected([X1,X2|Y]):-linked_to(X1,X2),is_connected([X2|Y]).
route_between(X, [Y,Z]):-begins_with(X,Y),ends_with(X,Z),is_connected(X).
```

第七章 专家系统

7.1.答：

（1）专家系统的定义

- 费根鲍姆（E．A．Feigenbaum）：“专家系统是一种智能的计算机程序，它运用知识和推理步骤来解决只有专家才能解决的复杂问题”
- 专家系统是基于知识的系统，用于在某种特定的领域中运用领域专家多年积累的经验和专门知识，求解需要专家才能解决的困难问题
  - *保存和大面积推广各种专家的宝贵知识*
  - *博采众长*

- *比人类专家更可靠，更灵活*

（2）专家系统的特点

①具有专家水平的专门知识

专家系统中的知识按其在问题求解中的作用可分为三个层次：数据级、知识库级和控制级

- 数据级知识（动态数据）：具体问题所提供的初始事实及在问题求解过程中所产生的中间结论、最终结论

数据级知识通常存放于数据库中

- 知识库级知识：专家的知识，这一类知识是构成专家系统的基础

一个系统性能高低取决于这种知识质量和数量

- 控制级知识（元知识）：关于如何运用前两种知识的知识

在问题求解中的搜索策略、推理方法

②能进行有效的推理

推理机构——能根据用户提供的已知事实，通过运用知识库中的知识，进行有效的推理，以实现问题的求解。*专家系统的核心是知识库和推理机*

③具有启发性

除能利用大量专业知识外，还必须利用经验判断知识来对求解问题作出多个假设（依据某些条件选定一个假设，使推理继续进行）

④ 能根据不确定（不精确）的知识进行推理

综合利用模糊的信息和知识进行推理，得出结论

⑤具有灵活性

知识库与推理机相互独立，使系统易于扩充，具有较大的灵活性

⑥具有透明性

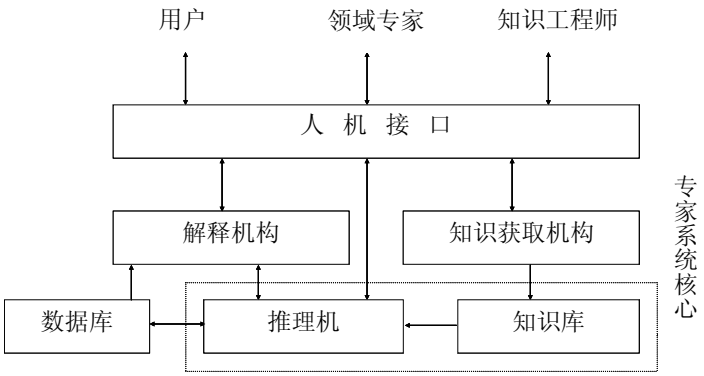
- 一般有解释机构，所以具有较好的透明性
- 解释机构向用户解释推理过程，回答“Why？”、“How？”等问题

⑦具有交互性

- 一般都为交互式系统，具有较好的人机界面
- 一方面它需要与领域专家或知识工程师进行对话以获取知识；另一方面它也需要不断地从用户处获得所需的已知事实并回答询问。

7.2.答：专家系统的一般结构

人机接口、推理机、知识库、动态数据库、知识获取机构、解释机构



知识库：主要用来存放领域专家提供的专门知识

(1) 知识表达方法的选择（*最多的三种表示方法是产生式规则、框架和语义网络*）

① 充分表示领域知识
② 能充分、有效地进行推理
③ 便于对知识的组织、维护与管理
④ 便于理解与实现

(2) 知识库管理冗余和矛盾一致性和完整性安全性

推理机

- 模拟领域专家的思维过程，控制并执行对问题的求解
- 能根据当前已知的事实，利用知识库中的知识，按一定的推理方法和控制策略进行推理，直到得出相应的结论为止
- 推理机包括推理方法和控制策略两部分
- 推理方法有精确推理和不精确推理*（已在推理章节介绍）*
- 控制策略主要指推理方向控制及推理规则选择策略
- 推理有正向推理、反向推理和正反向混合推理
- 推理策略一般还与搜索策略有关*（已在推理章节介绍）*
- 推理机性能/构造与知识的表示方法有关，但与知识的内容无关◊保证推理机与知识库的独立性，提高灵活性

知识获取机构

- “瓶颈”，是建造和设计专家系统的关键
- 基本任务是为专家系统获取知识，建立起健全、完善、有效的知识库，以满足求解领域问题的需要
- 要对知识进行一致性、完整性检测

人机接口

- 专家系统与领域专家、知识工程师、一般用户间进行交互的界面，由一组程序及相应的硬件组成，用于完成输入输出工作
- 更新、完善、扩充知识库；推理过程中人机交互；结束时显示结果　*内部表示形式与外部表示形式的转换*

数据库

- 又称“黑板”、“综合数据库”或“动态数据库”，主要用于存放用户提供的初始事实、问题描述及系统运行过程中得到的中间结果、最终结果等信息
- 数据库是推理机不可缺少的工作场地，同时由于它可记录推理过程中的各种有关信息，又为解释机构提供了回答用户咨询的依据*（需相应的数据库管理程序）*

解释机构：回答用户提出的问题，解释系统的推理过程，使系统对用户透明

7.3 答：

(1) 传统程序是依据某一确定的算法和数据结构来求解某一确定的问题，而专家系统是依据知识和推理来求解问题，这是专家系统与传统程序的最大区别。

传统程序 = 数据结构 + 算法

专家系统 = 知识 + 推理

(2) 传统程序把关于问题求解的知识隐含于程序中，而专家系统则将知识与运用知识的过程即推理机分离。*（使专家系统具有更大的灵活性，使系统易于修改）*

(3) 从处理对象来看，传统程序主要是面向数值计算和数据处理，而专家系统则面向符号处理。传统程序处理的数据多是精确的，对数据的检索是基于模式的布尔匹配，而专家系统处理的数据和知识大多是不精确的、模糊的，知识的模式匹配也多是不精确的。

(4) 传统程序一般不具有解释功能，而专家系统一般具有解释机构，可对自己的行为作出解

释。

(5) 传统程序因为是根据算法来求解问题，所以每次都能产生正确的答案，而专家系统则像人类专家那样工作，通常产生正确的答案，但有时也会产生错误的答案*（这也是专家系统存在的问题之一）*。专家系统有能力从错误中吸取教训，改进对某一工作的问题求解能力。

(6) 从系统的体系结构来看，传统程序与专家系统具有不同的结构。

7.4 答：可行性分析：威特曼（Watermam）从三方面研究如何选择适合专家系统开发的问题

(1)什么情况下开发专家系统是可能的? （满足!）

① 问题的求解主要依靠经验性知识，而不需要大量运用常识性知识

② 存在真正的领域专家，这也是开发专家系统最重要的要求之一

专家必须能够描述和解释他们用于解决领域问题的方法

③ 一般某领域中有多个专家，他们应该对领域答案的选择和精确度有基本一致的看法

④ 任务易，有明确的开发目标，且任务能被很好地理解

(2)什么情况下开发专家系统是合理的？（之一!）

① 问题的求解能带来较高的经济效益

② 人类专家奇缺，但又十分需要，且十分昂贵

③ 人类专家经验不断丢失

④ 危险场合需要专门知识

(3)什么情况下开发专家系统是合适的？（特征!）

① 本质——问题本质上必须能很自然地通过符号操作和符号结构来进行求解，且问题求解时需要使用启发式知识，需要使用经验规则才能得到答案

② 复杂性——问题不是太容易且较为重要

③ 范围——问题需要有适当的范围。选择适当的范围是专家系统的关键，一般有两个原则：一是所选任务的大小可驾驭；二是任务要有实用价值。

7.5 答：

专家系统的设计原则

(1)专门任务　　*领域大小*

(2)专家合作　　*反复磋商，团结协作*

(3)原型设计　　*从“最小系统”到“扩充式”开发*

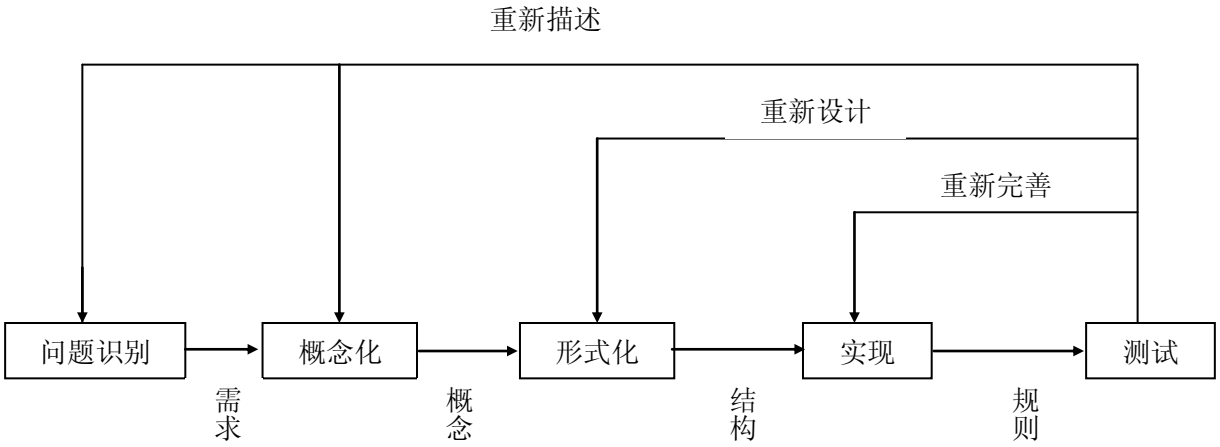
(4)用户参与　　*充实、完善知识库*

(5)辅助工具　　*提高设计效率*

(6)知识库与推理机分离　　*体现特征，灵活*

专家系统的开发步骤

知识工程比软件工程更强调渐进性、扩充性



(1) 问题识别阶段——知识工程师和专家确定问题的重要特点，抓住问题各主要方面的特征

① 确定人员和任务

② 问题识别：描述问题的特征及相应的知识结构，明确问题的类型和范围

③ 确定资源：确定知识源、时间、计算设备以及经费等资源

④ 确定目标：确定问题求解的目标

(2) 概念化阶段——主要任务是揭示描述问题所需的关键概念、关系和控制机制，子任务、策略和有关问题求解的约束

① 什么类型的数据有用，数据之间的关系如何？

② 问题求解时包括哪些过程，这些过程中有哪些约束？

③ 问题是如何划分成子问题的？

④ 信息流是什么？哪些信息是由用户提供的，哪些信息是应当导出的？

⑤ 问题求解的策略是什么？

(3)形式化阶段——把概念化阶段概括出来的关键概念、子问题和信息流特征形式化地表示出来

（究竟采用什么形式，要根据问题的性质选择适当的专家系统构造工具或适当的系统框架）

- 三个主要的因素是：

  假设空间　　基本的过程模型　　数据

形式化阶段假设空间

① 把概念描述成结构化的对象，还是处理成基本的实体？

② 概念之间的因果关系或时空关系是否重要，是否应当显式地表示出来？

③ 假设空间是否有限？

④ 假设空间是由预先确定的类型组成的，还是由某种过程生成的？

⑤ 是否应考虑假设的层次性？

⑥ 是否有与最终假设和中间假设相关的不确定性或其它的判定性因素？

⑦ 是否考虑不同的抽象级别？

形式化阶段基本的过程模型

- 找到可以用于产生解答的基本过程模型是形式化知识的重要一步
- 过程模型包括行为的和数学的模型

（如果专家使用一个简单的行为模型，对它进行分析，就能产生很多重要的概念和关系）

（数学模型可以提供附加的问题求解信息，或用于检查知识库中因果关系的一致性）

形式化阶段数据的性质

① 数据是不足的、充足的还是冗余的？
② 数据是否有不确定性？
③ 对数据的解释是否依赖于出现的次序？
④ 获取数据的代价是多少？
⑤ 数据是如何得到的？
⑥ 数据的可靠性和精确性如何？
⑦ 数据是一致的和完整的吗？

(4)实现阶段

- 把形式化知识变成计算机的软体，即要实现知识库、推理机、人机接口和解释系统*（知识的一致性和相容性）*
- 推理机应能模拟领域专家求解问题的思维过程和控制策略
- 必须很快地实现*（实现原型系统的目的之一是检查开发早期阶段的设计是否有效）*

(5)测试阶段

- 通过运行实例评价原型系统以及用于实现它的表达形式，从而发现知识库和推理机制的缺陷
- 性能不佳的因素：

① 输入输出特性，即数据获取与结论表示方面存在缺陷

例如，提问难于理解、含义模糊，使得存在错误或不充分的数据进入系统；结论过多或者太少，没有适当地组织和排序，或者详细的程度不适当

② 推理规则有错误、不一致或不完备
③ 控制策略问题，不是按专家采用的“自然顺序”解决问题

测试的主要内容：

① 可靠性——通过实例的求解，检查系统所得出的结论是否与已知结论一致
② 知识的一致性——向知识库输入一些不一致、冗余等有缺陷的知识，检查是否可检测出来

检查是否会给出不应给出的答案

检测获取知识的正确性*（如有某些自动获取知识功能）*

③ 运行效率——知识查询及推理方面的运行效率，找出薄弱环节及求解方法与策略方面的问题
④ 解释能力——一是检测能回答哪些问题，是否达到了要求；二是检测回答问题的质量*（说服力）*
⑤ 人机交互的便利性

7.6 答：

专家系统种类	解 决 的 问 题
解 释	根据感知数据推理情况描述
诊 断	根据观察结果推断系统是否有故障
预 测	推导给定情况可能产生的后果
设 计	根据给定要求进行相应的设计
规 划	设计动作
控 制	控制整个系统的行为

监　督	比较观察结果和期望结果
修　理	执行计划来实现规定的补救措施
教　学	诊断、调整、修改学生行为
调　试	建议故障的补救措施

(1) 解释型专家系统

- 能根据感知数据，经过分析、推理，从而给出相应解释。*（必须能处理不完全、甚至受到干扰的信息，给出一致且正确的解释）*
- 代表性：DENDRAL（化学结构说明）、PROSPECTOR（地质解释）等

(2) 诊断型专家系统

- 能根据取得的现象、数据或事实推断出系统是否有故障，并能找出产生故障的原因，给出排除故障的方案 *（目前开发、应用得最多的一类）*
- 代表性：PUFF（肺功能诊断系统）、PIP（肾脏病诊断系统）、DART（计算机硬件故障诊断系统）等

(3) 预测型专家系统

- 能根据过去和现在信息（数据和经验）来推断可能发生和出现的情况
  （天气预报、市场预测、人口预测等）

(4) 设计型专家系统

- 能根据给定要求进行相应的设计
  （工程设计、电路设计、服装设计）
- 代表性：XCON（计算机系统配置系统）、KBVLSI（VLSI 电路设计专家系统）等

(5) 规划型专家系统

- 能按给定目标拟定总体规划、行动计划、运筹优化等
  （机器人动作控制、军事规划、城市规划等）
- 代表性：NOAH（机器人规划系统）、SECS（帮助化学家制定有机合成规划的专家系统）、TATR（帮助空军制订攻击敌方机场计划的专家系统）等

(6) 控制型专家系统

- 能根据具体情况，控制整个系统的行为
- 代表性：YES/MVS（帮助监控和控制 MVS 操作系统）

(7) 监督型专家系统

- 能完成实时的监测任务，并根据监测到的现象作出相应的分析和处理
- 代表性：REACTOR（帮助操作人员检测和处理核反应堆事故）

(8) 修理型专家系统

- 能根据故障的特点制订纠错方案，并能实施该方案排除故障，当制订的方案失效或部分失效时，能及时采取相应的补救措施

(9) 教学型专家系统

- 能根据学生学习过程中所产生的问题进行分析、评价、找出错误原因，有针对性地确定教学内容或采取其它有效的教学手段
- 代表性：GUIDON（讲授有关细菌感染性疾病方面的医学知识）

(10) 调试型专家系统

- 能根据相应的标准检测被测试对象存在的错误，并能从多种纠错方案中选出适用于当前情况的最佳方案，排除错误

专家系统的**应用领域**已扩展到数学、物理、化学、医学、地质、气象、农业、法律、教育、交通运输、机械、艺术以及计算机科学本身，甚至渗透到政治、经济、军事等重大决策部门，产生了巨大的社会效益和经济效益，同时也促进了人工智能基本理论和基本技术的发

展。

7.7 答：（1）正向推理：见教材 P206 图 7.7

（2）反向推理：见教材 P212 图 7.12

7.8 答：（1）知识获取的任务

- 基本任务：为专家系统获取知识，建立起健全、完善、有效的知识库，以满足求解领域问题需要

①抽取知识　*识别、理解、筛选、归纳等，及自学习*

②知识的转换

- *第一步：从专家及文献资料处抽取的知识转换为某种知识表示模式，如产生式规则、框架等　（知识工程师完成）*
- *第二步：该模式表示的知识转换为系统可直接利用的内部形式。（输入及编译实现）*

③知识的输入　　*知识编辑器*

④知识的检测　　*不一致、不完整等*

⑵知识获取的模式

①非自动知识获取(人工移植)*知识工程师 知识编辑器*

②自动知识获取

- 系统具有获取知识的能力，它不仅可以直接与领域专家对话，从专家提供的原始信息中学习到专家系统所需的知识，而且还能从系统自身的运行实践中总结、归纳出新的知识，发现知识中可能存在的错误，不断自我完善，建立起性能优良、知识完善的知识库
  - ➢具有识别语音、文字、图像的能力
  - ➢具有理解、分析、归纳的能力
  - ➢具有从运行实践中学习的能力

③半自动知识获取

7.9 答：

正确性

(1)系统设计的正确性

① 系统设计思想的正确性　　*如目标、原则等*

② 系统设计方法的正确性　　*如知识表达方法、知识推理方法、控制策略、解释方法等*

③ 设计开发工具的正确性　　*如正确使用和正确维护*

(2)系统测试的正确性

① 测试目的、方法、条件的正确性

② 测试结果、数据、记录的正确性

(3)系统运行的正确性

① 推理结论、求解结果、咨询建议的正确性

② 推理解释及可信度估算的正确性

③ 知识库知识的正确性　　*语法、语义和语用及专业内容*

有用性

(1)推理结论、求解结果、咨询建议的有用性

(2)系统的知识水平、可用范围、易扩充性、易更新性等

(3)问题的求解能力（解题速度、推理效率），可能场合和环境

(4)人机交互的友好性

(5)运行可靠性、易维护性、可移植性

(6)系统的经济性*（软硬件投资、运行维护费用、设计开发费用和系统运行取得的直接或间接经济效益）*

7.10 答：（1）四种主要的类型：

① 用于开发专家系统的程序设计语言

② 骨架系统

③ 通用型知识表达语言

④ 专家系统开发环境

（2）专家系统开发环境（工具包）

- AGE 是斯坦福大学研制的一个专家系统开发环境。
- AGE 是典型的模块组合式开发工具，为用户提供了一个通用的专家系统结构框架，并将该框架分解为许多在功能和结构上较为独立的的组件部件，这些组件已预先编制成标准模块存在系统中。
- AGE 采用了黑板模型来构造专家系统结构框架。
- 可通过两条途径构造自己的专家系统：

① 用户使用 AGE 现有的各种组件作为构造材料，很方便地来组合设计自己所需的系统。

② 用户通过 AGE 的工具界面，定义和设计各种所需的组成部件，以构造自己的专家系统。

- *应用 AGE 已经开发了一些专家系统，主要用于医疗诊断、密码翻译、军事科学等方面。*

7.11 答：

EMYCIN 是由 MYCIN 系统抽去原有的医学领域知识，保留骨架而形成的系统*（产生式规则表达知识、目标驱动的反向推理控制策略）*。

- EMYCIN 具有 MYCIN 的全部功能：

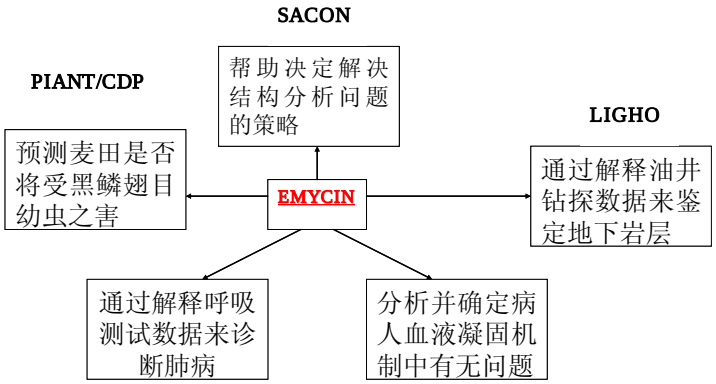
① 解释程序——可以向用户解释推理过程。

② 知识编辑程序及类英语的简化会话语言——提供一开发知识库的环境，使得开发者可以使用比 LISP 更接近自然语言的规则语言来表示知识。

③ 知识库管理和维护手段——所提供的开发知识库的环境还可以在进行知识编辑及输入时进行语法、一致性、是否矛盾和包含等检查。

④ 跟踪和调试功能

EMYCIN 开发的一些专家系统（适合开发各种领域咨询、诊断型专家系统）。



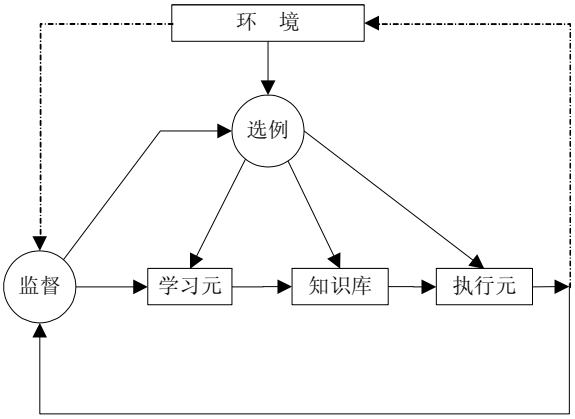
第八章 机器学习

8.2 答：

（1）学习是一项复杂的智能活动，学习过程与推理过程是紧密相连的　　学习中所用的推理越多，系统的能力越强

（2）机器学习是一门研究机器获取新知识和新技能，并识别现有知识的学问　*“机器”——计算机（电子，以后还可能是中子计算机、光子计算机或神经计算机等）*

8.3 答：机器学习系统的结构及基本功能



当监督环节为示教人时，为示教式学习系统；当监督环节为监督器时，为自学式学习系统。

① 知识库　存储（记忆）、积累知识

• 长期记忆（LTM）　*先验知识背景　如事物的基本概念和定义、定律和公理，博弈的基本规则等*

• 中期记忆（MTM）　*环境事物的各种具体知识*

• 短期记忆（STM）　*环境变化的信息和数据　　事实库或“黑板*

② 学习元　学习系统的核心环节

• 采集环境信息息　*选例环节或直接采集*

• 接受监督指导　　*监督环节的示教、指导信息或评价准则*

• 进行学习推理　　*获得有关问题的解答和结论*

• 修改知识库　　*将推理结果输入知识库，对知识增删改*

③ 执行元　识别、论证、决策、判定

模式分类器、专家咨询解释系统、智能控制机构、机械手/人等

如执行元行动结果直接引起环境的变化◊ “在线”学习系统

机器人规划、生产过程控制、机器博弈等

④ 监督环节人:示教者;监督器:评价准则或检验标准

• 工作执行效果评价——*接受来自执行元环节的反馈信息，对系统的工作执行效果进行评价和检验*

• 制定评价标准——*接受来自环境变化的信息，制定和修订评价标准和检验标准*

• 监督学习环节——*根据评价和检验的结果，对学习环节进行示教、训练或指导*

• 控制选例环节——*根据环境变化信息及工作执行效果的反馈，控制选例环节，选取其它事例或样本*

⑤ 选例环节

作用是从环境中选取有典型意义的事例或样本，作为系统的训练集或学习对象。如挑选

典型病历，以便提高学习效率，加速学习过程。选例环节可以由人或机器来实现

⑥ 环境

系统获取知识和信息的来源，执行的对象和人物等。如，医疗专家系统的病员、病历档案、医生、诊断书等；模式识别系统的文字、图象、物景；博弈系统的对手、棋局；智能控制系统的被控对象和生产过程等。

8.4 答：

（1）. 机械学习模型

- 机械学习——一种最简单的机器学习方法
- 机械学习是最基本的学习过程，任何学习系统都必须记住它们获取的知识
- 机械学习系统：知识的获取是以较为稳定和直接的方式进行的，不需要系统进行过多的加工
- 机械学习就是记忆，即把新的知识存储起来，供需要时检索调用，而不需要计算和推理
- 归纳过程可以简化成推导过程

直接使用求根公式计算一个一元二次方程的根　　自学

（2）. 机械学习的主要问题

① 存储组织信息

② 环境的稳定性与存储信息的适用性问题。*密切监视外界环境的变化，不断地更换所保存的信息；核对。*

③ 存储与计算之间的权衡。*预估算；“选择忘却”技术*

8.5 答：

（1）示例学习：

病态细胞的分类识别例如图，

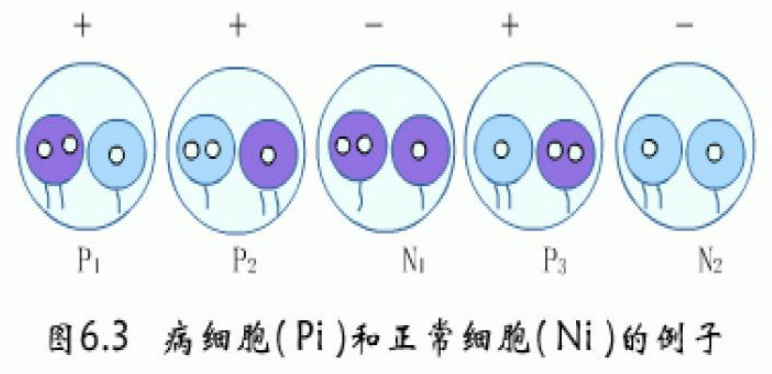


图6.3　病细胞(Pi)和正常细胞(Ni)的例子

- 正例——三个病细胞(P1, P2, P3)，
- 反例——二个正常细胞(N1, N2)；
- 每个细胞由二个细胞体组成
  * 细胞体表示为三元组：（核数、尾数、染色状），
  * P1：{(2, 2, 深) (1, 1, 浅)}。
- 学习任务——从例子集中归纳出有病状 X 的细胞

***A.* 概念描述的搜索和获取**

概念描述

• 假设不必给每个特性（属性）都指明应取值：

* 没有给出值的特性（以？指示）——对于该概念的描述无关紧要；

* 病细胞假设(a)：{(2, ?, ?) (?, 1, 深)}，一个细胞体有二个胞核；另一个有一个尾巴，且染色是深的。

• 病细胞假设空间的半序图（图 6.5）

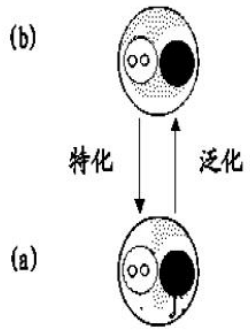


图6.4 假设空间上的一个泛化/特化关系

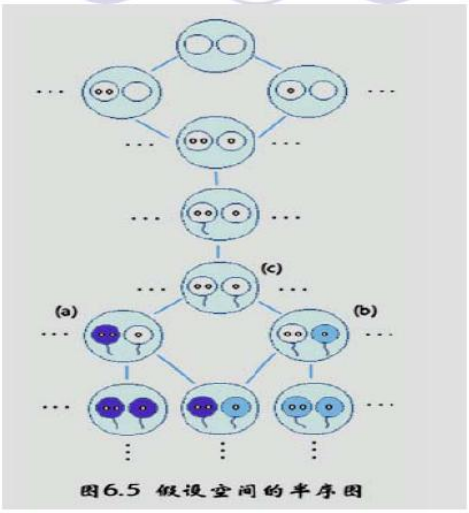


图6.5 假设空间的半序图

• 假设之间的关系弧指示泛化/特化关系，

• 假设空间上的一个泛化/特化关系（图 6.4），

* 假设（ b ）不考虑细胞体是否有尾巴，比假设（ a ）复盖更多的例子；

* 假设（ b ）比假设（ a ）泛化；

* 假设（ a ）比假设（ b ）特化。

• 底层假设——最特化（具体）的概念描述：

* 所有特性都给定特别值， 对应于例子空间中的一个例子。

• 顶层假设——最泛化的概念描述：

* 不指定任何具体的特性值，

* 表示为{(? ? ?)，(? ? ?)}。

• 假设空间中的搜索方式

• 特化搜索——从最泛化的假设（概念描述）出发，每次取用一个新的例子，就产生一些特化的描述，直到将初始最泛化的假设特化为解描述。

•泛化搜索——从最特化的假设（相应于例子空间中的一个例子）开始，每次取用一个新的例子时，就产生一些泛化的描述，直到产生出足够泛化的解描述。

• 大多数示例学习方法都采用这二种方法或这二个方法的结合。

B. 逐步泛化的学习策略

• 采用宽度优先、自底向上的搜索方式：

• 将第一个正例（P1）作为初始假设（H1）——极端特化的假设；

• 正例（P2）用于指导系统生成泛化的假设（H2和H3）：

* 多个泛化的假设——不同的映射会导致不同的假设，- 假设H1中包含了二个对象（细胞体）；

* 采用保守原则——最低限度的泛化，：- 新的假设刚好覆盖现有的假设/例子。

• 反例（N1）用来剪裁过于泛化的假设：

- H3是过于泛化的假设，因为其蕴涵了反例N1。

基本策略：

- 遇见正例就泛化某些假设以保证假设的完全描述性，
- 遇见反例则删去某些假设以保证假设的一致描述性，
- 直至得到一个既完全又一致的解描述(假设)为止。
- 这个解描述作为满足给定例子集的概念定义——学习系统获得的新知识。

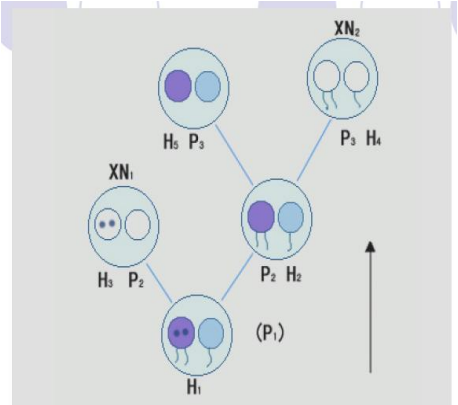


图6.6 逐步泛化学习策略的假设空间搜索

C. 逐步特化的学习策略

- 采用宽度优先、自顶向下的搜索方式（与泛化策略相反）：
- 新例子的加入会导致新假设的增加和已存在假设的删除（与泛化策略类似）。
- 正例和反例所起的作用与泛化策略相反：
- 反例——生成一些特化假设；

＊ 采用保守的原则——最低限度的特化：- 新的假设在覆盖已有正例的同时只是刚好能排斥反例；

- 正例——剪裁过于特化的假设。

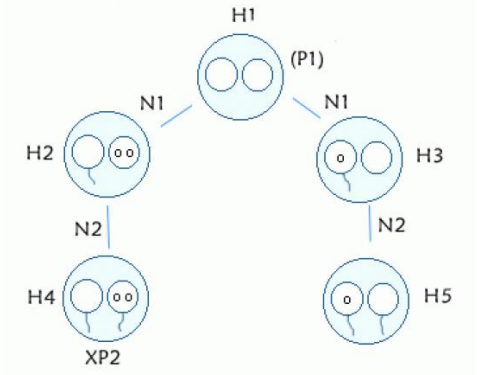


图6.8 逐步特化学习策略的假设空间搜索

（2）类比学习：

“肖锋象部消防车”中，考虑肖锋与消防车之间的相似

- 关于肖锋（*目标框架*）与消防车（*源框架*）的框架为：

肖锋	是一个（ISA）	人
	性别	男
	活动级	
	音量	
	进取心	中等

消防车	是一辆（ISA）	车辆
	颜色	红
	活动级	快
	音量	极高
	燃料效率	中等
	梯高	异或（长，短）
进取心	是一种（ISA）	个人品德

- 现在目的是用消防车的信息来扩充肖锋的内容

启发式规则：

① 选择那些用极值填写的槽
② 选择那些已知为重要的槽
③ 选择那些与源框架没有密切关系的槽
④ 选择那些填充值与源框架没有密切关系的槽
⑤ 使用源框架中的一切槽

- 这组规则用来寻找一种好的传递。对上述例题，将有下面一些结果：

① 活动级槽和音量级槽填有极值，所以它们首先入选
② 如果它们不存在，那么根据第②规则选择那些标记为特别重要的槽　　*本例无此情况*
③ 下一规则将选择梯高槽，因为该槽不出现在其它类型的车辆中
④ 下一规则将选颜色槽，因其它车辆都不是红色
⑤ 最后一条规则，若用它，则消防车的所有槽均为可能的相似

8.6-8.8 答：略

第九章 人工神经网络

9.1 答：

（1）误差纠正学习；Δwkj = ηek(n)xj(n)；*yk*(*n*)为输入 *xk*(*n*)时，神经元 *k* 在 *n* 时刻的实际输出，*dk*(*n*)表示应有的输出（可由训练样本给出）；其中η 为学习步长，这就是通常所说的误差纠正学习规则（或称 delta 学习规则）。

（2）Hebb 学习；Δwkj(n) = F(yk(n),xj(n))；当某一突触（连接）两端的神经元同步激活（同为激活或同为抑制）时，该连接的强度应为增强，反之应减弱；由于Δ *wkj* 与 *yk*(*n*)，*xj*(*n*)的相关成比例，有时称为相关学习规则。

（3）竞争（Competitive）学习；

$$\Delta w_{ji}=\begin{cases}\eta\left(x_i-w_{ji}\right), & \text{若神经元 } j \text{ 竞争获胜}\\ 0, & \text{若神经元 } j \text{ 竞争失败；}\end{cases}$$

在竞争学习时，网络各输出单元互相竞争，最后达到只有一个最强者激活，最常见的一种情况是输出神经元之间有侧向抑制性连接，这样原来输出单元中如有某一单元较强，则它将获胜并抑制其它单元，最后只有此强者处于激活状态。

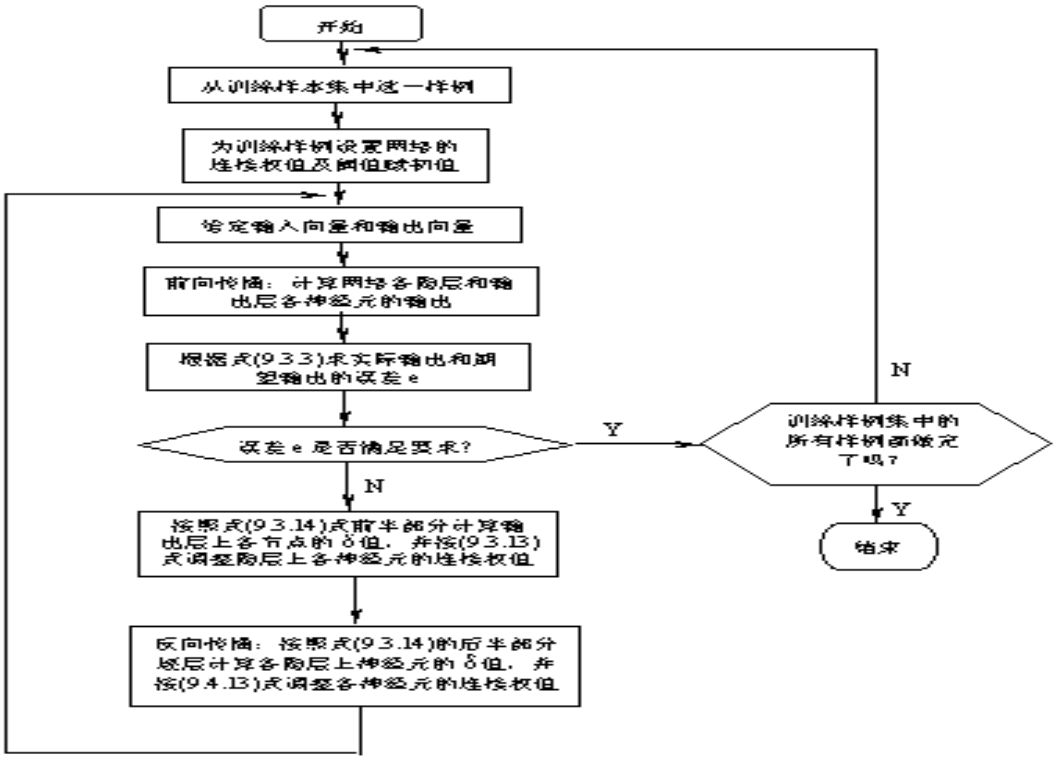
9.2 答：略

9.3 答：B-P 算法的学习过程如下：

（1）选择一组训练样例，每一个样例由输入信息和期望的输出结果两部分组成。

（2）从训练样例集中取一样例，把输入信息输入到网络中。

（3）分别计算经神经元处理后的各层节点的输出。
（4）计算网络的实际输出和期望输出的误差。
（5）从输出层反向计算到第一个隐层，并按照某种能使误差向减小方向发展的原则，调整网络中各神经元的连接权值。
（6）对训练样例集中的每一个样例重复（3）—（5）的步骤，直到对整个训练样例集的误差达到要求时为止。



文件头:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "UTLab.h"
//BP 神经网络
#define n1 7     //输入节点数
#define n2 11    //中间层节点个数
#define n3 5     //输出节点数
#define m 400    //学习次数
void initialization(){}
float bpnet(float tension, float inthick,float outthick, float meng, float tan,float ping,float tu)
{
    float in1[n1];
    float out1[n2];
    float out2[n3];
    float w1[n2][n1];
    float w2[n3][n2];
    float a1[n2];
```

```
float a2[n3];
float t[n3];
float df1[n2];
float df2[n3];
float s2[n3];
float s1[n2];
float w[n2][n3];
float z[n2][n3];
float a=0.5;
int i=0;
int j=0;
int k=0;
int n=0;
in1[1]=tension;
in1[2]=inthick;
in1[3]=outthick;
in1[4]=meng;
in1[5]=tan;
in1[6]=ping;
in1[7]=tu;
initialization();
for(n=1;n<=m;k++){
    for(i=1;i<=n2;i++){
        for(j=1;j<=n1;j++){
            a1[j]=a1[j]+w1[i][j]*in1[j];
            out1[j]=sigmoid(a1[j]);
        }
    }
    for(i=1;i<=n3;i++){
        for(j=1;j<=n2;j++){
            a2[j]=a2[j]+w2[i][j]
```

文件尾:

```
            *in1[j];
            out2[j]=a2[j];
        }
    }
    for(i=1;i<=n3;i++){                    //计算敏感度,更新权值
        df2[i]=1;
        s2[i]=-2*(t[i]-out2[i]);
        for(j=1;j<=n2;j++){
        w2[i][j]=w2[i][j]+a*out1[j];
        }
    }
    for(i=1;i<=n3;i++){
```

```
            for(j=1;j<=n2;j++){
            w[j][i]=w2[i][j];
            }
        }
        for(j=1;j<=n2;j++){
            df1[j]=(1-out1[j])*out1[j];
            s1[j]=0;
            for(i=1;i<=n3;i++){
                z[j][i]=df1[j]*w[j][i];
                s1[j]=s1[j]+z[j][i]*s2[i];
            }
        }
        for(j=1;j<=n2;j++){
            for(k=1;k<=n1;k++){
                w1[j][k]=w1[j][k]+a*s1[j]*in1[k];
            }
        }
    }
    return out2[n3];
}
void main(){}
```

9.4 答：略

9.5 答：Hopfield 网络是一个单层回归自联想记忆网络。

将记忆用能量函数（Lyapunov）来表示，并且在每个处理单元（神经元）中引入了异步工作方式。显著特点就是：许多已有的结构，不管是信息处理单元还是信息存储单元，都是并行工作的。但从信息处理的角度看，虽支持并行分布信息存储，但是信息处理并不是真的以并行方式进行。优点是：通常的神经网络要求在所有的时间里，每一个处理单元都要有整个网络的全局信息，而 Hopfield 网络就减少了这种要求，Hopfield 联想记忆模型迭代过程为不断地从一顶角转向具有更小能量的另一顶角，直至稳定于具有最小能量的一个顶角与之相对应，连续的 Hopfield 联想记忆模型迭代过程，从一个起始状态到另一个全局极小值（预存状态）来穿越超空间当所有的神经元都被访问到，并且没有神经元再改变状态时，就认为网络已经收敛。

9.6 答：

下面的程序描述了 Hopfield 神经网络求解 TSP 的改进算法，其中 A、D 为实验值.

算法

0. 置 $t=0, A=1.5, D=1$； 1. 读入 N 城市之间的距离 $d_{xy}(x,y=1,2\cdots,n)$ 文件.

2. 计算神经元之间的权重 $T_{xi,yj}$ 和输入偏置 I_{xi}：

$T_{xi,yj}=-A\delta_{xy}-A\delta_{ij}-Dd_{xy}\delta_{j,i-1},(x,y,i,j=1,2,\cdots,n),I_{xi}=2A$

3. $u_{xi}(t)$ 的初始值在 0 附近随机产生$(x,i=1,2,\cdots,N)$.

4. 计算 $V_{xi}(t)(x,i=1,2,\cdots,n),V_{xi}(t)=1/2(1+\tanh(u_{xi}(t)/u_0))$，这里 $u_0=0.02$

5. 利用神经元动态方程，计算 $\Delta u_{xi}(t),(x,i=1,2,\cdots,n).\Delta u_{xi}(t)=\sum_{y=1}^{n}\sum_{j=1}^{n}T_{xi,yj}V_{yj}+I_{xi}.$

6. 根据一阶尤拉法计算 $u_{xi}(t+1),(x,i=1,2,\cdots,n),u_{xi}(t+1)=u_{xi}(t)+\Delta u_{xi}(t)\Delta(t)$，这里 Δt 取 0.5.

7. 如果系统达到平衡状态，那么终止程序，否则返回第 4 步.

用 pascal 写的 hopfield 神经网络解决 TSP 问题的代码：

```
initialize
    dist[max_city,max_city] := 0.0 ;
    FOR c1 := 'A' TO max_city DO
     FOR i := 1 TO max_position DO
       u[c1,i] := u00 + (((2 * random - 1.0) / 10.0) * u0) ;
    clrscr ;
    writeln('TSP    [c] 1987 Knowledge Garden Inc.') ;
    writeln(' 473A Malden Bridge Rd') ;
    writeln(' Nassau, NY 12123') ;
    writeln ;
    writeln('Press <Space Bar> to begin - Press again to stop iterating.') ;
    read(kbd,ch) ;
  END ; (* initialize *)

 BEGIN
  initialize ;
  iterate ;
 END.
```

9.7 答：Kohonen 网络自组织映射算法的实现过程如下：

①定义一个输入向量与神经元接收到反馈信息加权值之间的相似性度量

②选定输入向量与输出神经元所有连结权之间的相似度度量最大的为获胜输出单元

③更新获胜神经元的权值和获胜神经元周围的邻域中所有单元的权值

④不断自动地调整连接权最后由稳定的优胜神经元产生输出

第十章 人工智能游戏

10.1 答：

电脑游戏中的人工智能是人类技能的一个人工版本,在游戏中我们把人工智能解释为非玩家角色(NPC)也称为机器智能。当前人工智能游戏研发还处于初级阶段。过去几年所用到的一些常用的技术，如脚本行为和 A* 路径搜索算法已经和二维图像技术相比了,一些公司

开始尝试从人工智能领域发展出更高级的技术(如决策树和强化学习)。许多游戏研发人员认为仿生机器人是处理人工智能 NPC 最合理的手段,仿生机器人本质上是生活在虚拟环境下的综合创造物, 可以通过多种学习算法来适应环境。

10.2 答：略