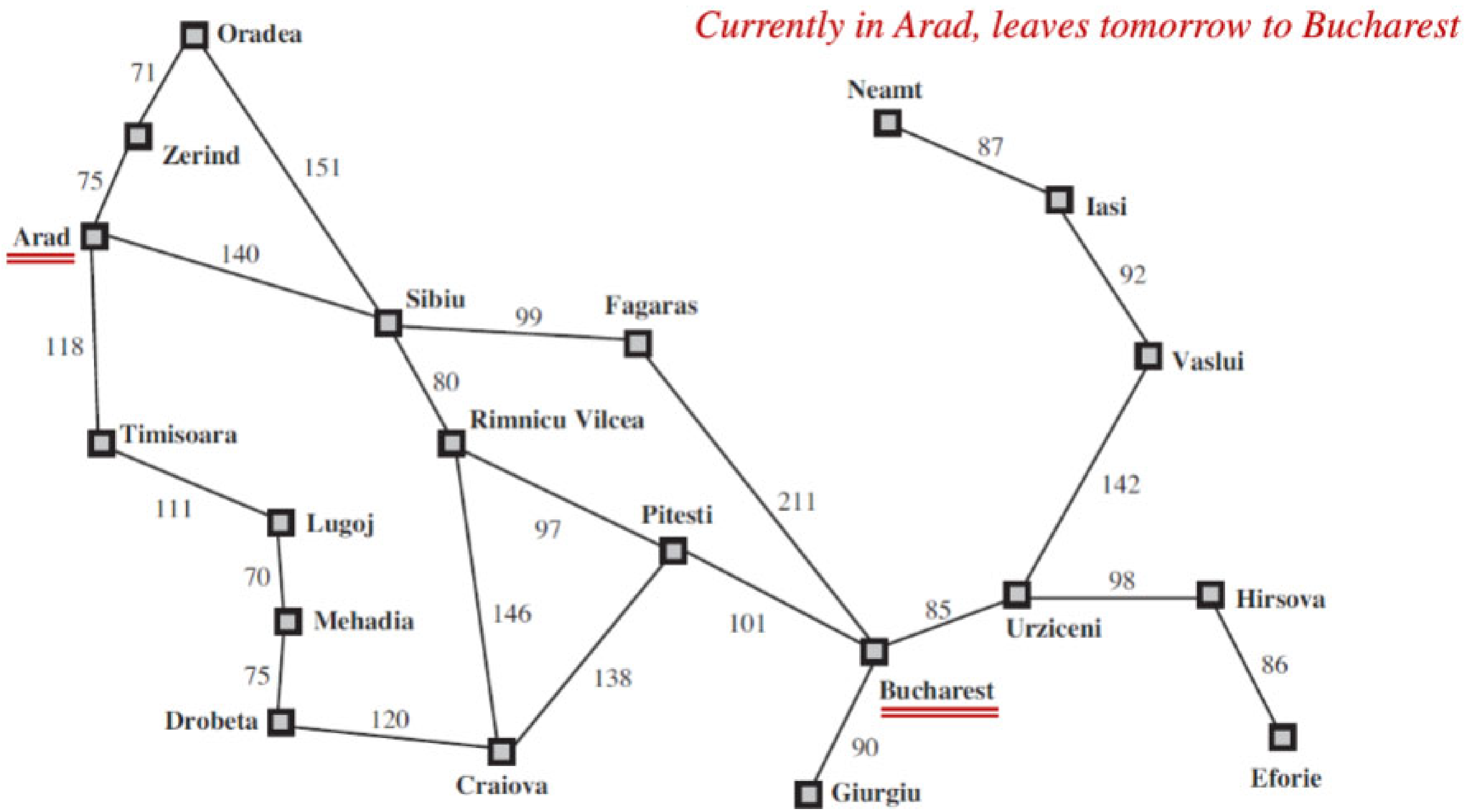
第二章　用搜索求解问题的基本原理

AI中每个研究领域都有其各自的特点和规律,但就求解问题的过程看,都可抽象为一个问题求解过程。

问题求解(Problem-solving)是AI领域中的一大课题，它涉及归约、推断、决策、规划、常识推理、定理证明等相关过程的核心概念，是人工智能中研究得较早而且比较成熟的一个领域。

一个实际的问题



- 目标的形式化
- 问题的形式化

  给定目标下确定需要考虑哪些行动和状态。
- 搜索
- 执行

几个概念

- **The Solution 解**

  是一个达到目标的动作序列

- **The process 过程**

  寻找该动作序列，称其为搜索

- **Problem formulation 问题形式化**

  给定一个目标，决定要考虑的动作与状态

- **Why search 为何搜索**

  对于某些NP全问题，只能通过搜索来解决

问题形式化的五个要素

- **Initial State 初始状态** $In(Arad)$
- **Action 动作**

  ACTION(s) 返回s状态下可执行的动作序列。

  $\{Go(Zerind), Go(Sibiu), Go(Timisoara)\}$

- **Transition Model 转换模型**

  RESULT(s, a) 返回在s下动作a之后的状 态。

  RESULT$(In(Arad), Go(Zerind)) = In(Zerind)$

- **Goal Test 目标测试**

  确定一个给定的状态是否是目标状态。$\{In(Bucharest)\}$

- **Path Cost 路径代价** 即每条路径所分配的一个数值代价。 $c(s, a, s')$.

相关术语

■ 状态空间

问题的状态空间可以形式化地定义为：初始状态、动作和转换模型。

■ 图

状态空间形成一个图，其中节点表示状态、链接表示动作。

■ 路径

状态空间的一条路径是由一系列动作连接的一个状态序列。

问题求解

- 任何问题求解技术都包括两个重要的方面:**表示和搜索**
- 表示是基本的,搜索必须要在表示的基础上进行。
- 求解一个问题主要包括三个阶段：目标表示、搜索和执行。

2.1搜索求解问题的基本思路

很多问题的求解方法都是采用试探的搜索方法，在一个可能的解空间中寻找一个满意解。

为模拟这些试探性的问题求解过程而发展的一种技术就称为**搜索**。

智力游戏

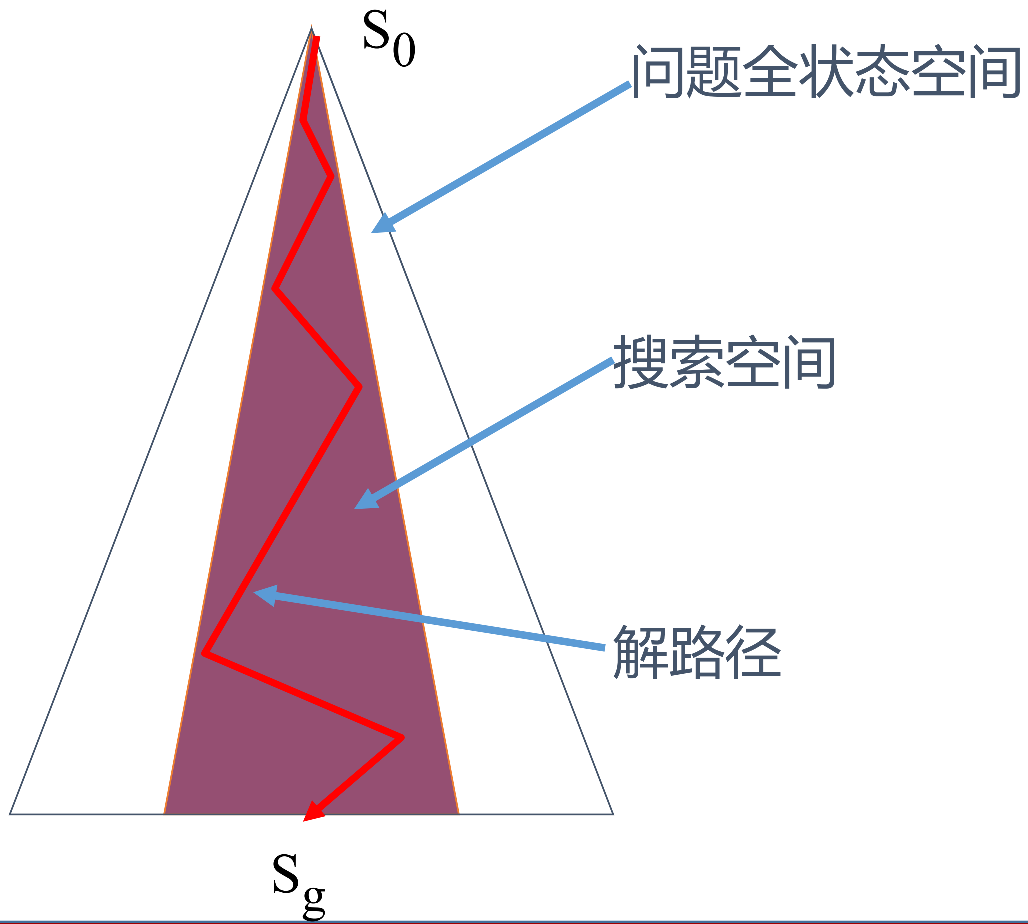
找到解决方法后：

1.这个方案所用的步骤是否最少?

2.如果不是，如何能找到最优的方案?

3.计算机上如何实现这样的搜索?

搜索



搜索问题的含义：如何在一个比较大的问题空间中，只搜索比较小的范围，就找到问题的解。

搜索

- 搜索什么：通常指的就是目标。
- 在哪里搜索：就是状态空间，状态空间通常是一系列状态的汇集。

人工智能中的搜索可以分成两个阶段：

- 状态空间的生成阶段
- 在该状态空间中对所求解问题状态的搜索

搜索的方式

根据是否使用启发式信息分为：

- **盲目搜索**：按预定的搜索策略进行搜索。这种搜索具有很大的盲目性，效率不高，不便于复杂问题的求解。

- **启发式搜索**：在搜索过程中加入了与问题有关的启发性信息，用于指导搜索朝着最有希望的方向前进，加速问题的求解并找到最优解。

显然，盲目搜索不如启发式搜索的效率高，但是启发式搜索需要的信息很少，或者根本没有，或者很难抽取，所以盲目搜索还是很重要的搜索策略。

2.2 实现搜索过程的三大要素

- 搜索对象：在什么之上进行搜索
- 搜索的扩展规则：如何控制从一种状态转化成另一种状态，使得搜索得以前进
- 搜索的目标测试：指搜索在什么条件下终止

状态空间法

- 把可能的解都表示为一个状态，也就是要把要求解问题的各个方面抽象成计算机可以理解的方式并存储起来。
- 这个过程必须做到把所有和解决问题相关的信息全部保留，存储这些信息的数据结构被叫做状态空间。

状态

状态（state）是为描述某些不同事物间的差别而引入的一组最少变量q_0，q_1，q_2…，q_n的有序集合，并表示为：

$$Q = (q_0, q_1, \cdots, q_n)$$

其中，每个元素q_i称为状态变量。给定每个分量的一组值，就得到一个具体的状态。

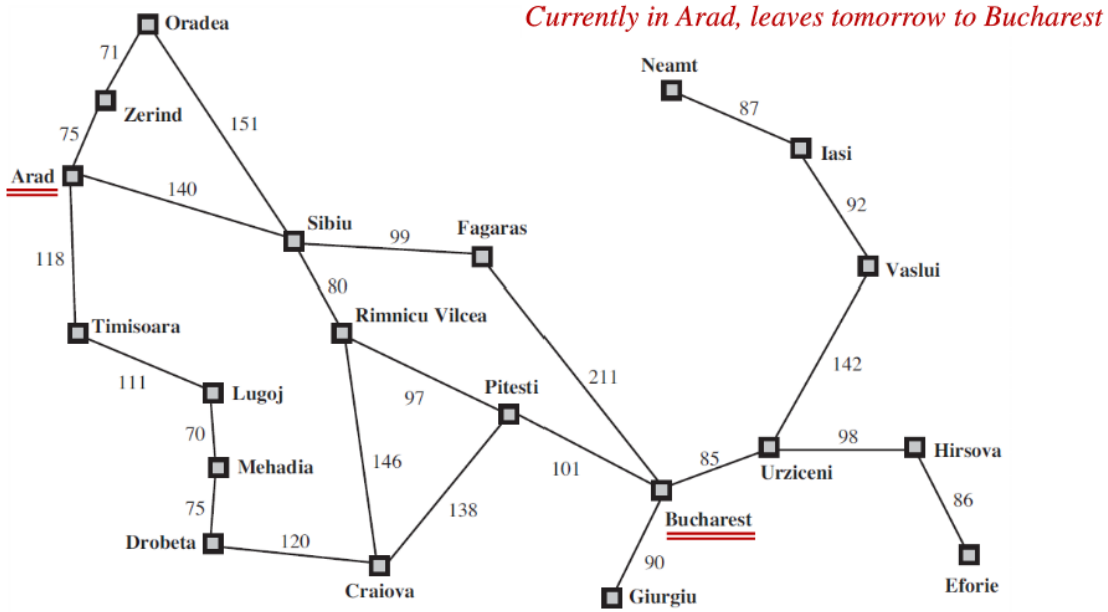
状态的表示还可以根据具体应用，采取灵活的方式确定数据结构，如二维数组、树形结构等。

状态空间

问题的状态空间是一个表示该问题全部可能状态及其关系的集合，通常以图的形式出现，图上的节点对应问题的状态，节点之间的边对应的是状态转移的可能性，边上的权可以对应转移所需的代价。

问题的解：

- 图中的一个状态
- 从开始状态到某个状态的一条路径
- 达到目标所花费的代价



扩展规则

控制策略：节点的扩展顺序选择、算子的选择、数据的维护、搜索中回路的判定、目标测试等。

生成系统：约束条件，算子

目标测试

- 是否满足所有限制条件
- 是不是目标

2.3 通过搜索求解问题

定义状态空间：

初始状态集合S

操作符集合F

目标状态集合G

因此，可把状态空间记为三元组（S，F，G）。

问题状态空间法的基本思想

（1）将问题中的已知条件看成状态空间中**初始状态**；将问题中要求的目标看成状态空间中**目标状态**；将问题中其它可能的情况看成状态空间的**任一状态**。

（2）设法在状态空间**寻找一条路径**，由初始状态出发，能够沿着这条路径达到目标状态。

搜索求解问题的基本步骤

（1）根据问题，定义出相应的状态空间，确定出状态的一般表示，它含有相关对象的各种可能的排列。

(2) 规定一组操作(算子)，能够使状态从一个状态变为另一个状态。

(3) 决定一种搜索策略，使得能够从初始状态出发，沿某个路径达到目标状态。

$$S_0 \xrightarrow{O_1} S_1 \xrightarrow{O_2} S_2 \xrightarrow{O_3} \cdots \xrightarrow{O_k} G$$

$O_1, \cdots, O_k$ ：状态空间的一个解。

八数码问题的状态空间。

2	3	1
5		8
4	6	7

初始状态

1	2	3
8		4
7	6	5

目标状态

状态集 S ：所有摆法

操作算子：

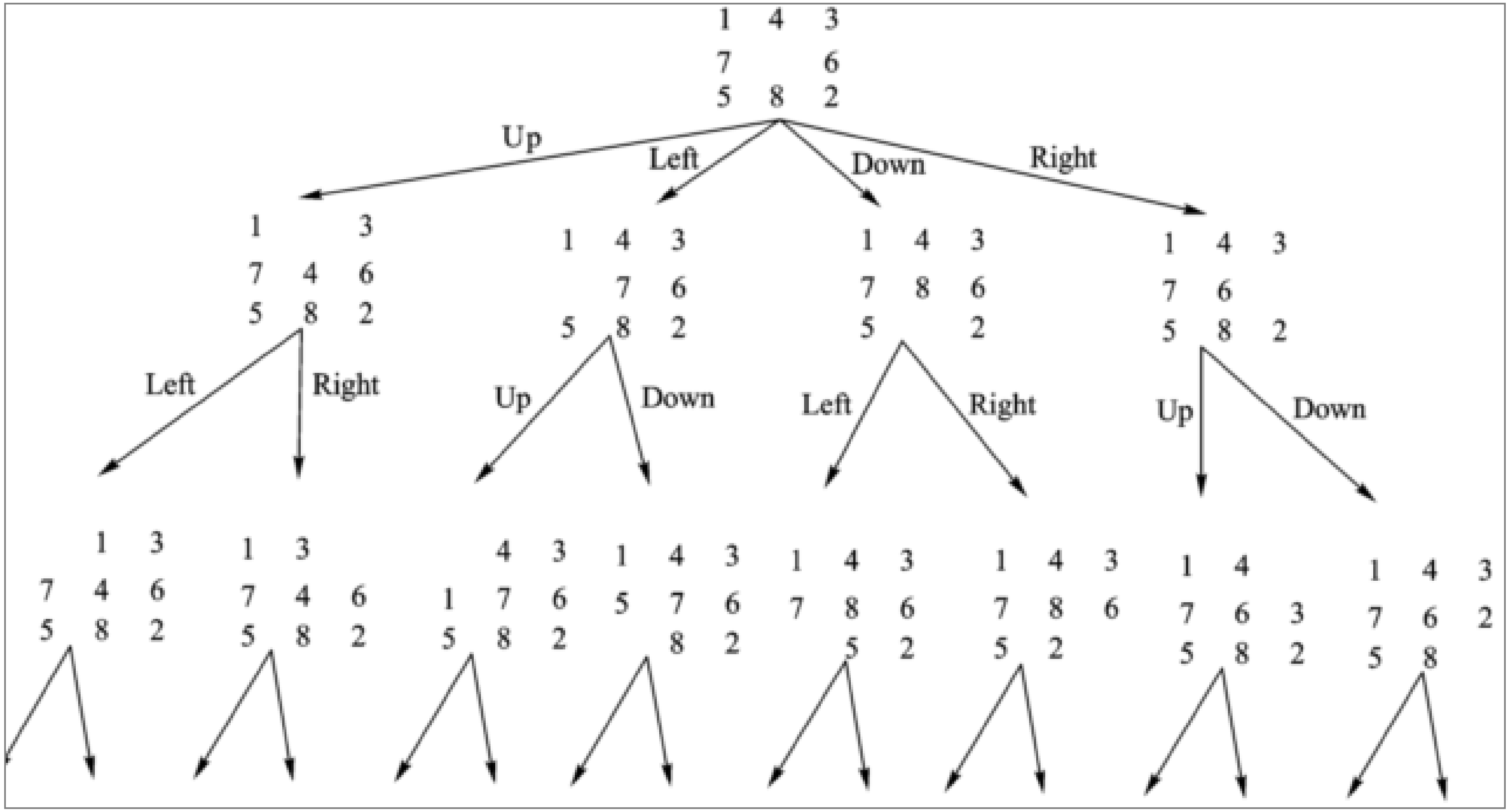
将空格向上移Up

将空格向左移Left

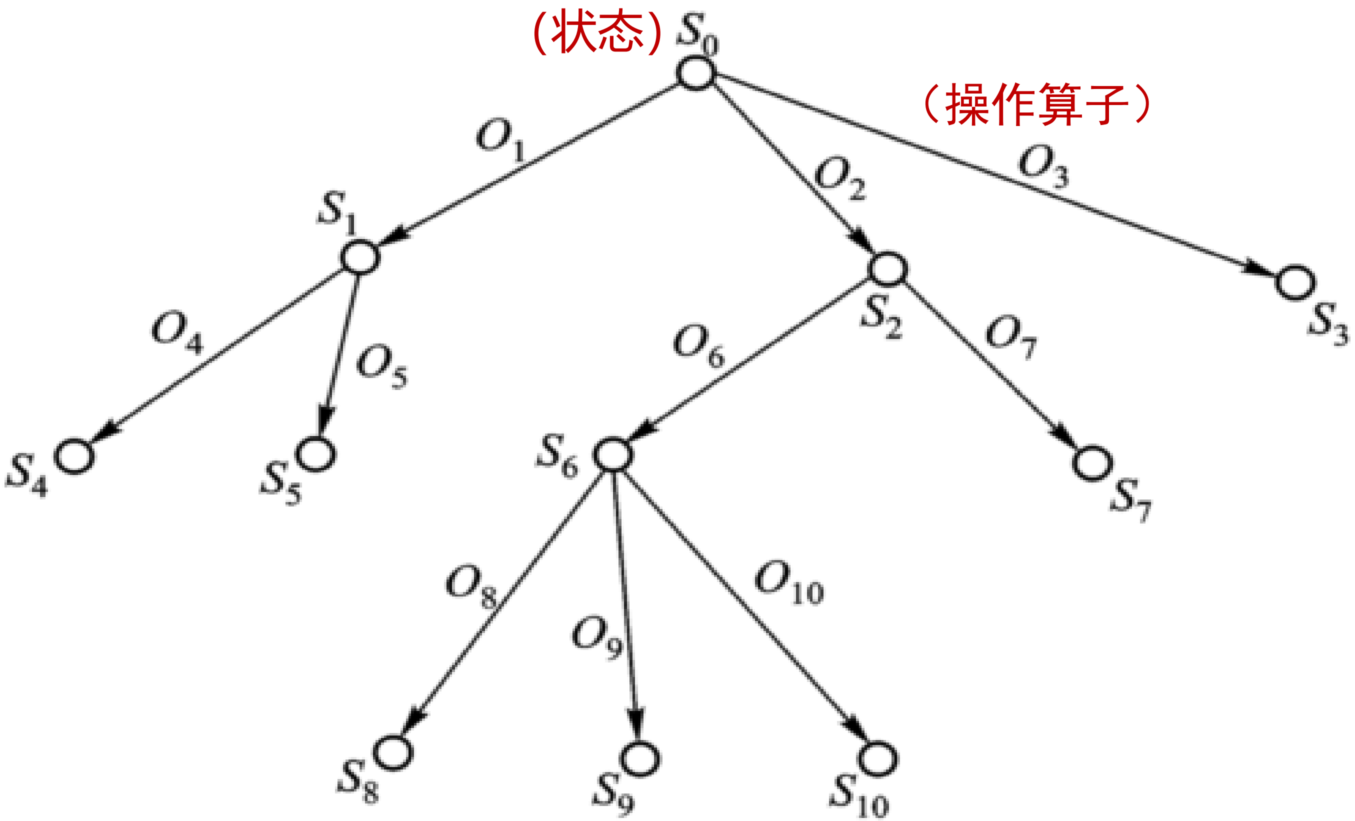
将空格向下移Down

将空格向右移Right

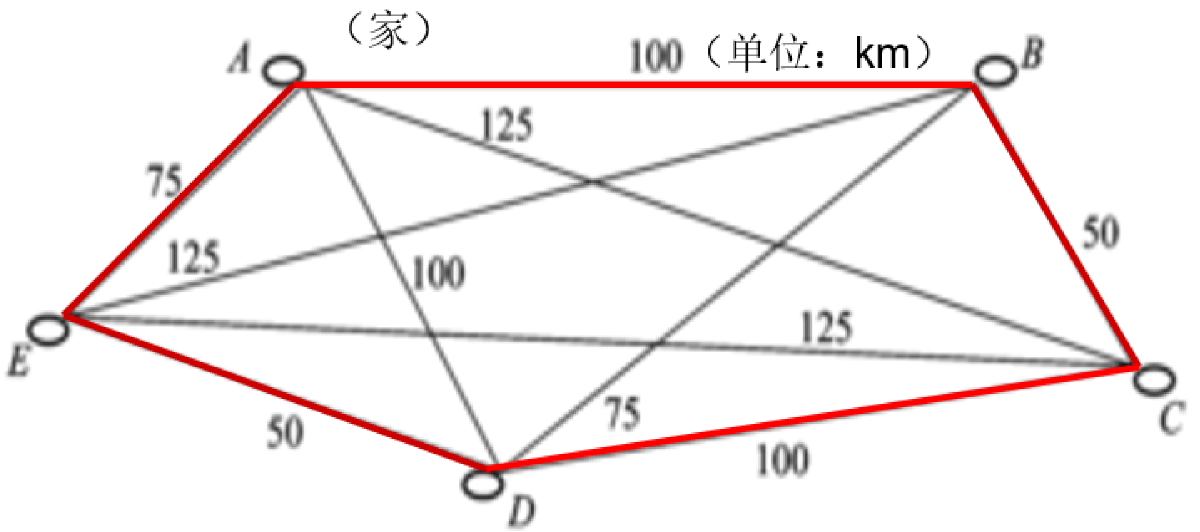
八数码状态空间图


1 4 3
7 6
5 8 2
Up
Left
Down
Right
1 3
7 4 6
5 8 2
1 4 3
7 6
5 8 2
1 4 3
7 8 6
5 2
1 4 3
7 6
5 8 2
Left
Right
Up
Down
Left
Right
Up
Down
1 3
7 4 6
5 8 2
1 3
7 4 6
5 8 2
4 3
1 7 6
5 8 2
1 4 3
5 7 6
8 2
1 4 3
7 8 6
5 2
1 4 3
7 8 6
5 2
1 4
7 6 3
5 8 2
1 4 3
7 6 2
5 8

状态空间的有向图描述

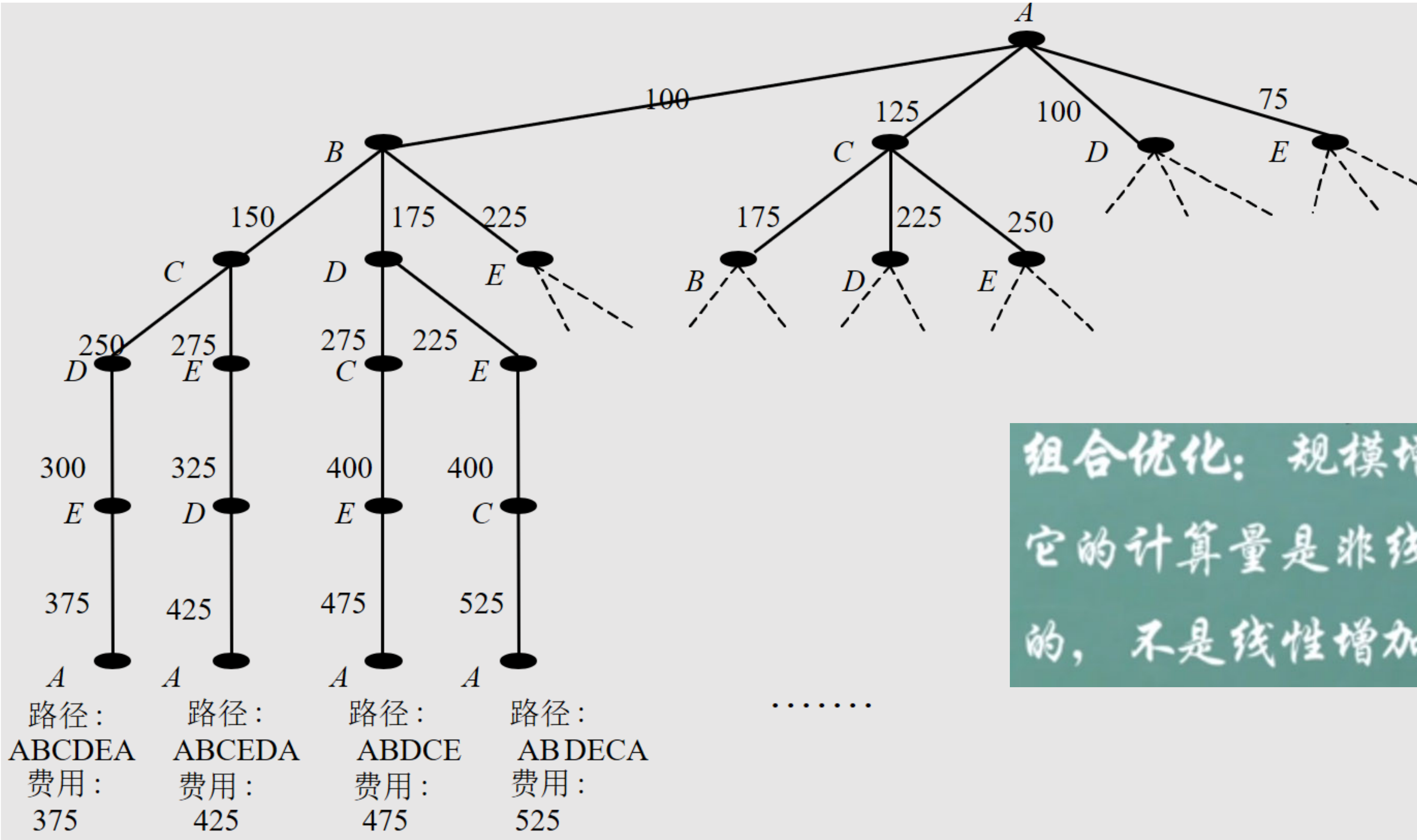

(状态) S_0
(操作算子)
O_1
O_2
O_3
S_1
S_2
S_3
O_4
O_5
O_6
O_7
S_4
S_5
S_6
S_7
O_8
O_9
O_{10}
S_8
S_9
S_{10}

旅行商问题（traveling salesman problem, TSP）



可能路径：费用为375的路径（*A*，*B*，*C*，*D*，*E*，*A*）

旅行商状态空间图（部分）



组合优化：规模增加一点，它的计算量是非线性增加的，不是线性增加的。

水壶问题

给定两个水壶，一个可装4加仑水，一个能装3加仑水。水壶上没有任何度量标记。有一水泵可用来往壶中装水。

问：怎样在能装4加仑的水壶里恰好只装2加仑水？

1. 定义状态空间

可将问题进行抽象，用数偶(x,y)表示状态空间的任一状态。

x — 表示4gallon水壶中所装的水量，x=0，1，2，3或4；

y — 表示3gallon水壶中所装的水量，y=0，1，2或3；

初始状态为 (0, 0)

目标状态为 (2, ?)

? 表示水量不限，因为问题中未规定在3加仑水壶里装多少水。

2. 确定一组操作

r1（X,Y | X<4）→（4,Y）

4加仑水壶不满时，将其装满；

r2（X,Y | Y<3）→（X,3）

3加仑水壶不满时，将其装满；

r3 (X,Y) | X>0）→（X-D,Y）

从4升水壶里倒出一些水；

r4 (X,Y) | Y>0）→（X,Y-D）

从3升水壶里倒出一些水；

r5（X,Y | X>0）→（0,Y）

把4加仑水壶中的水全部倒出；

r6（X,Y | Y>0）→（X，0） 把3加仑水壶中的水全部倒出；

r7 (X,Y | X+Y≥4∧Y>0) → (4,Y-(4-X)) 把3加仑水壶中的水往4加仑水壶里倒，直至4加仑水壶装满为止

r8 （X,Y | X+Y≥3∧X>0）→（X-（3-Y）, 3）

把4加仑水壶中的水往3加仑水壶里倒，直至3加仑水壶装满为止；

r9 （X,Y | X+Y≤4∧Y>0）→（X+Y, 0）

把3加仑水壶中的水全部倒进4加仑水壶里；

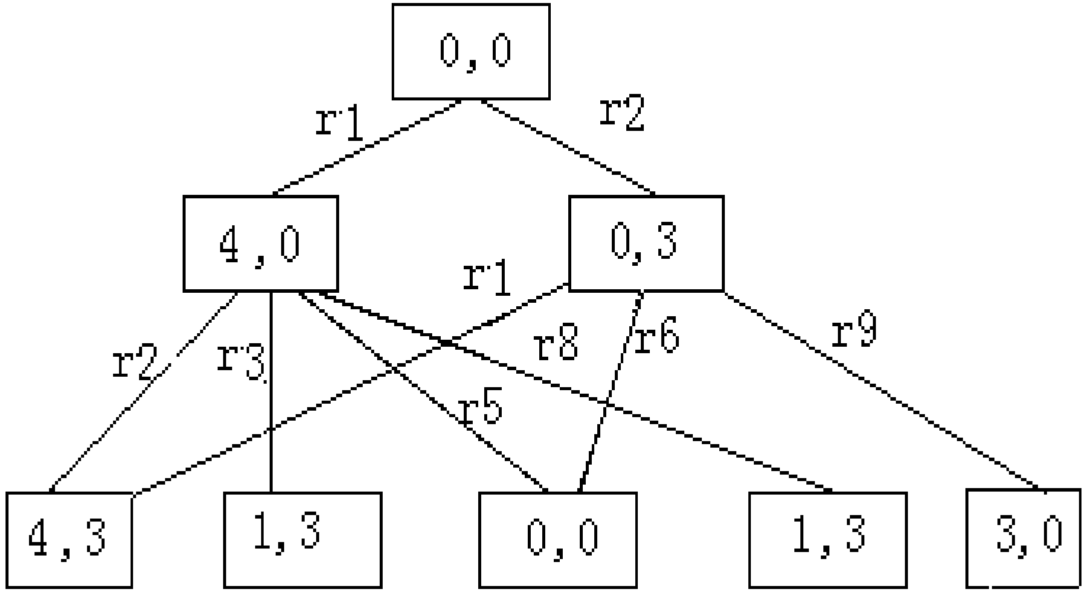
r10 （X,Y | X+Y≤3∧X>0）→（0, X+Y）

把4加仑水壶中的水全部倒进3加仑水壶里；

3. 选择一种搜索策略

该策略为一个简单的循环控制结构：

选择其左部匹配当前状态的某条规则，并按照该规则右部的行为对此状态作适当改变，然后检查改变后的状态是否为某一目标状态，若不是，则继续该循环。

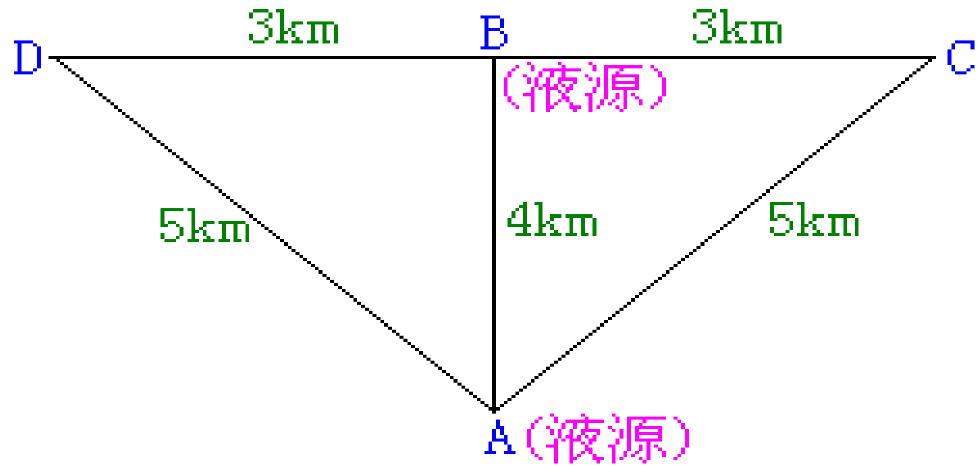


水壶问题的搜索路径

4加仑水壶中含水加仑数	3加仑水壶中含水加仑数	所应用的规则
0	0	初始状态
0	3	r2
3	0	r9
3	3	r2
4	2	r7
0	2	r5
2	0	r9

分配问题

有两个液源A和B。A的流量为100L/m ，B的流量为50L/m。现要求它们以75L/m的流量分别供应两个同样的洗涤槽C，D。液体从液源经过最大输出能力为75L/m的管道进行分配，A、B、C、D的位置、距离如图所示。并且要求只允许管子在液源或洗涤槽位置有接头。问：如何连接管子使得管材最少？



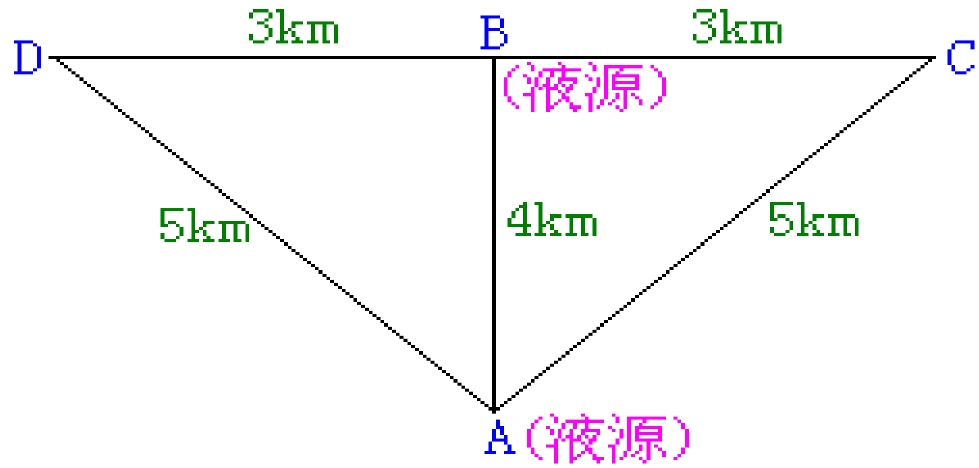
1. 定义状态空间中的状态表示

状态的表示形式表示为：

（A=? B=? C=? D=? ）

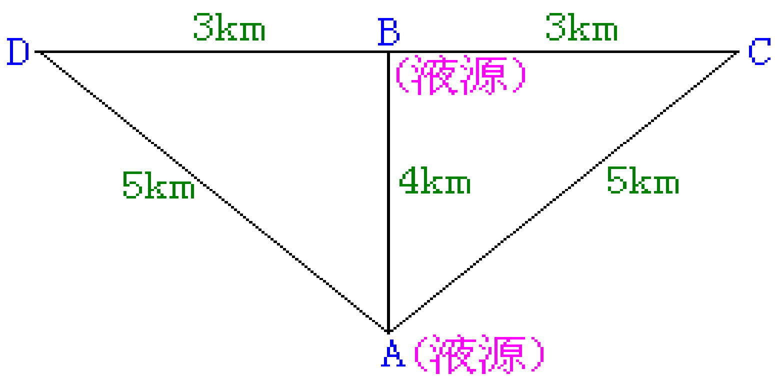
初 态：（A=100 B=50 C=0 D=0 ）

目标状态：（A=0 B=0 C=75 D=75）



2 定义操作

现在取从一处到另一处流量的增量，为各点流量与各处所需流量的最大公约数(great common divisor)。100、50、75的GCD为25，所以取增量为25。



	A	B	C	D
A	×			
B	×	×		
C	⊗	⊗	×	
D	⊗	⊗		×

①本身到本身不必传递，用×表示；②洗涤槽不必往液源送，用⊗表示

可行操作

1 A➞B (A⩾75∧B<75)➞(A-25, B+25, C,D)

4km

2 A➞C (A⩾25∧C<75)➞(A-25, B, C+25)

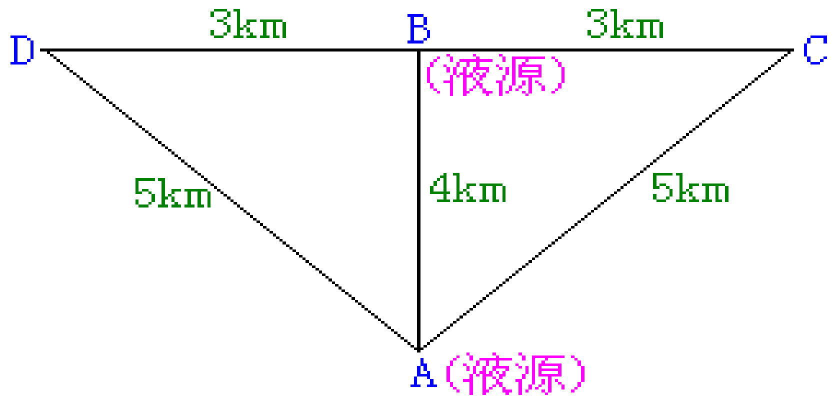
5km

3 A➞D (A⩾25∧D<75)➞(A-25, B, C, D+25)

5km

4 B➞C (B⩾25∧C<75)➞(A, B-25, C+25, D)

3km

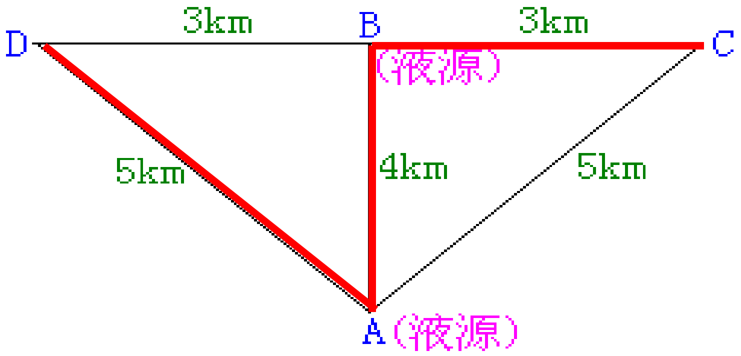


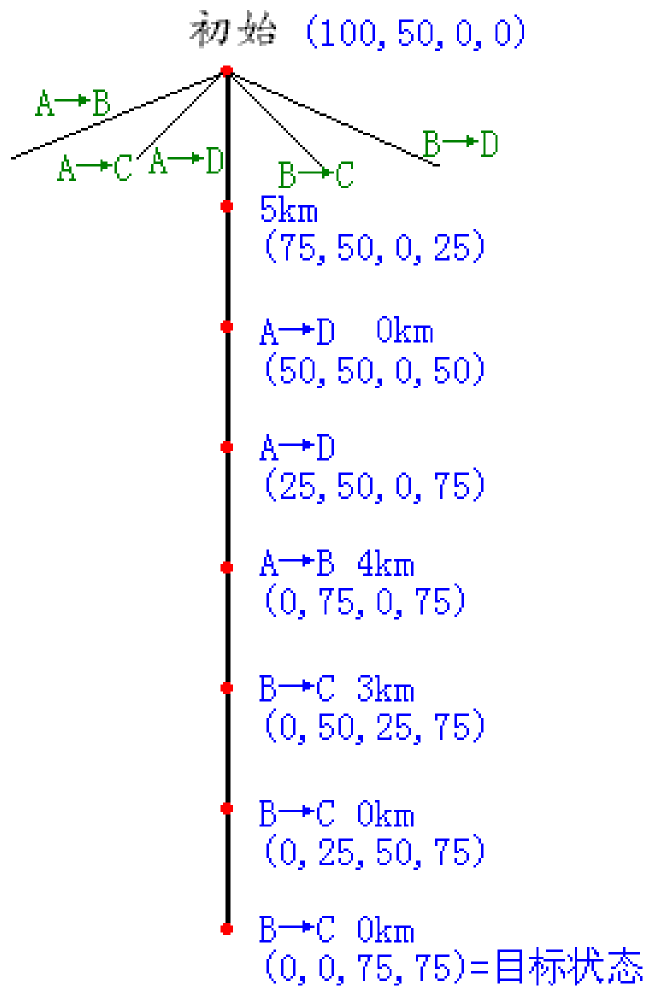
定义策略

采用耗尽式搜索，即每次将7个操作试用一遍。对于该具体问题，搜索时要注意：

① 若操作重复时，只算一次距离；

② 边搜索边求出距离最短的管长。





2.4　问题特征分析

一般要考虑：

- 问题可分解成为一组独立的、更小、更容易解决的子问题吗？
- 当结果表明解题步骤不合适的时候，能忽略或撤回吗？
- 问题的全域可预测吗？
- 在未与所有其它可能解作比较之前，能说当前的解是最好的吗
- 用于求解问题的知识库是相容的吗？
- 求解问题一定需要大量的知识吗？或者说，有大量知识时候，搜索时应加以限制吗？
- 只把问题交给电脑，电脑就能返回答案吗？或者说，为得到问题的解，需要人机交互吗？

问题的可分解性

如果问题能分解成若干子问题，则将子问题解出后，原问题的解也就求出来了。人们称这种解决问题的方法为**问题的归约**。

问题规约的基本思想

将大问题或复杂问题分解为小的或简单的本原问题集合。

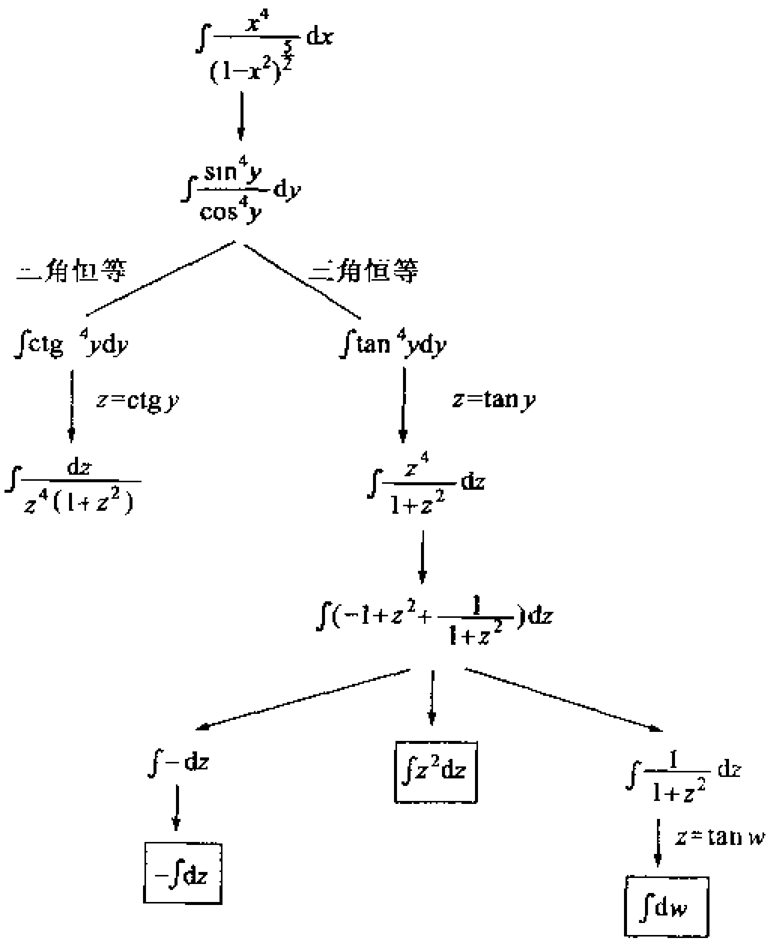
问题规约的搜索过程就是在状态空间中，以待求解的原始问题为出发点，以本原问题为目标，不断搜寻子问题的过程。

符号积分

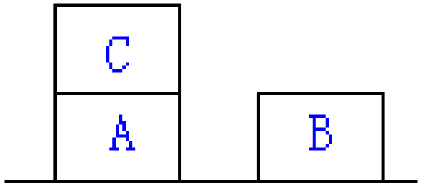
原始问题：求解不定积分 $\int \frac{x^4}{(1-x^2)^{5/2}} dx$

本原问题：$\int dx$

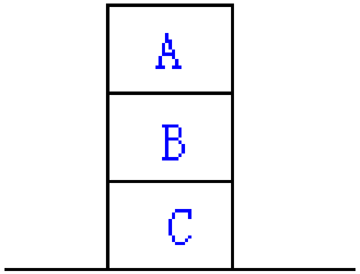
算子：积分规则



积木问题——机器人规划的抽象模型



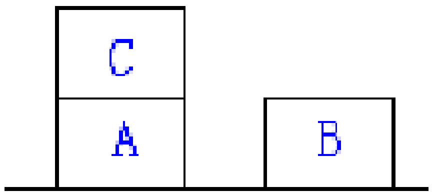
已知



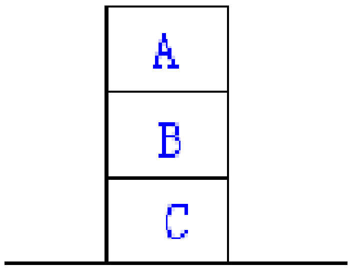
求的目标

积木问题关心的是积木块的相对位置：某一积木在桌上或某一积木在另一积木上。机器人只能一次拿一块积木，每次搬动时积木上面必须是空的。

积木的相对位置可用谓词表示



已知



求的目标

初始状态： ontabel(B)∧clear(B)∧
ontabel(A)∧on(C,A)∧clear(C)

目标状态： ontabel(C)∧on(B, C)∧on(A, B)

其中目标状态可分解为：

子问题1: ontabel(c)

子问题2: on(B, C)

子问题3: on(A, B)

积木问题

机器人所需完成的操作：

OP1： clear(x)→ontabel(x)

无论x在何处，若x上无物体，则可将x放于桌上

OP2： clear(x)∧clear(y)→On(x, y)

若x，y上无物体，则可将x放在y上

求解方法

- 一种方法是采用全面搜索的方法；
- 一种是用分解子问题的方法。

从目标来看，总问题可分解成三个子问题，但这三个子问题不能按任意次序求解。

子问题1: ontabel(c)

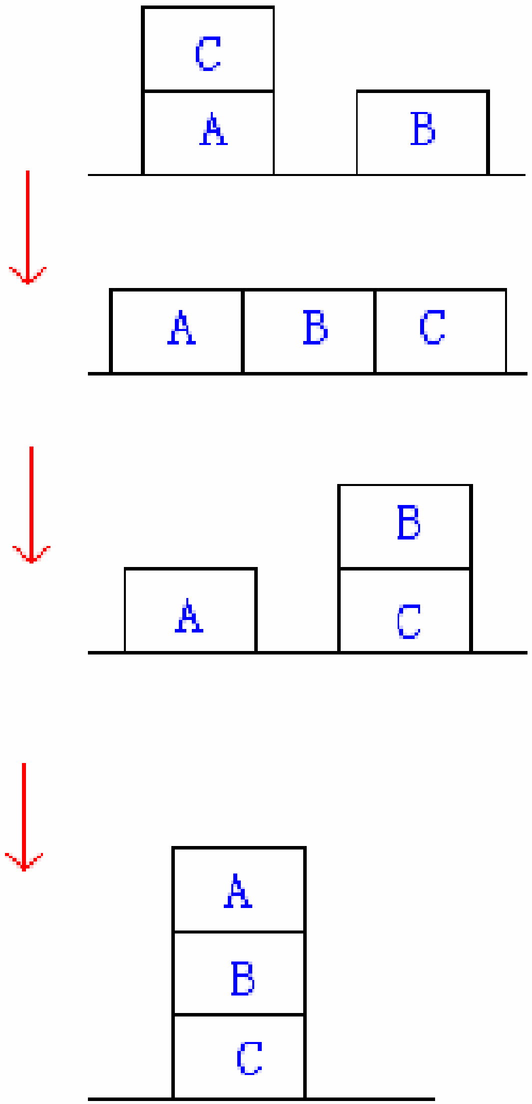
初态　子问题2: on(B, C)

子问题3: on(A, B)

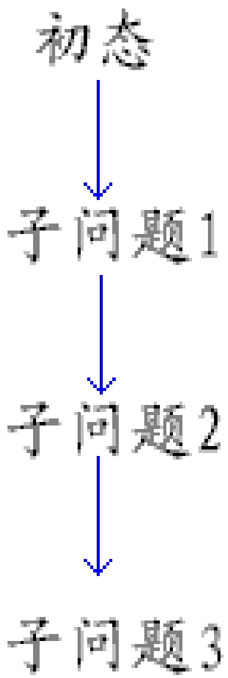
求解子问题1 （积木全部放在桌上）

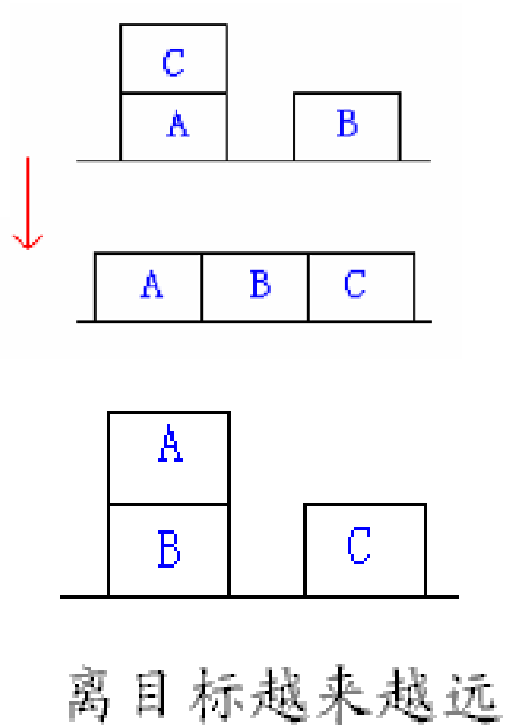
求解子问题2 （B放C上）

求解子问题3 （A放B上）



但若从初态出发，将on(A, B)作为子问题1 首先求解，这样会使搜索离目标越来越远。



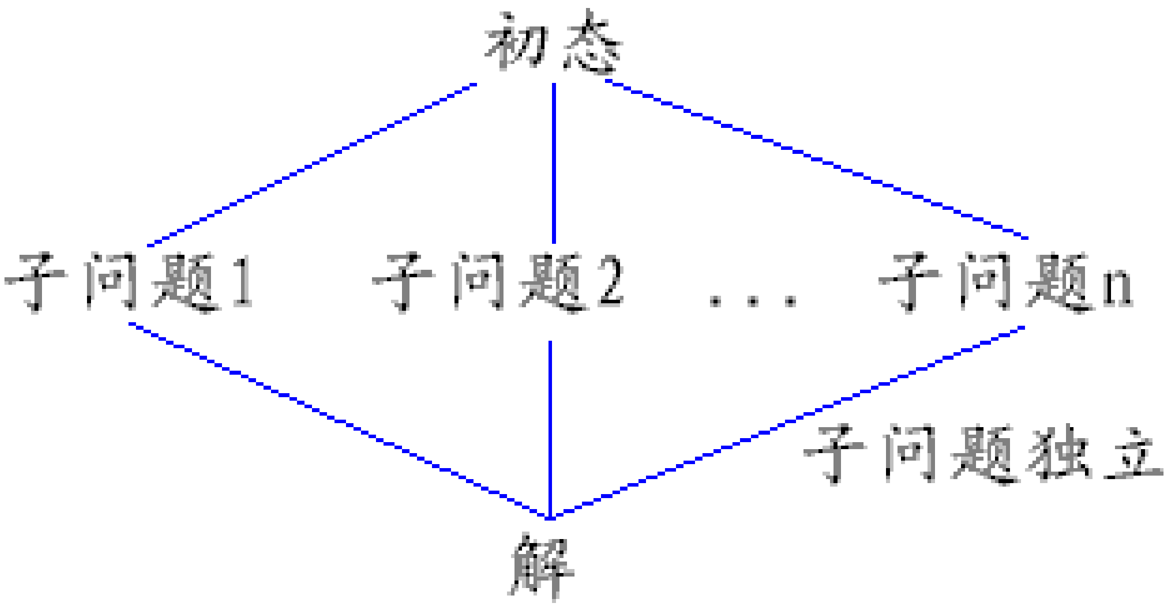


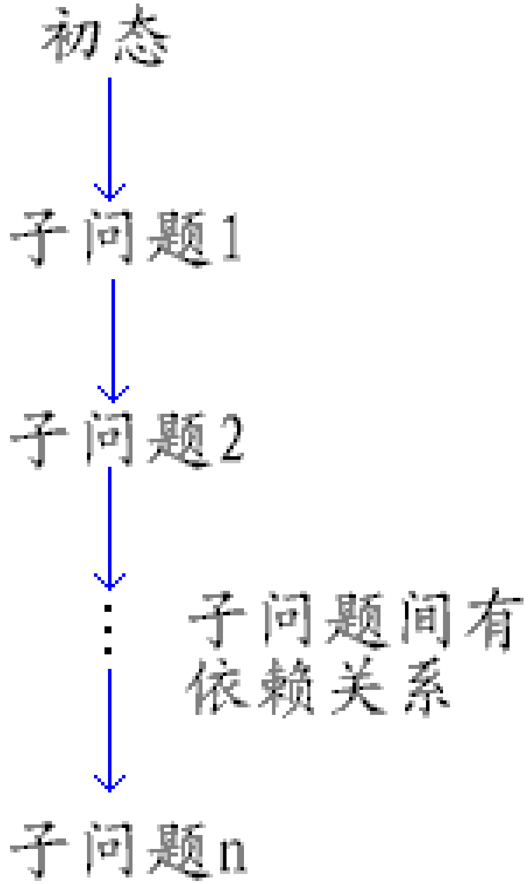
子问题1: ontabel(c)

子问题2: on(B, C)

子问题3: on(A, B)

子问题之间的关系





问题求解步骤是否可撤回

- 求解步骤可忽略

  搜索控制结构不需要带回溯

- 可复原

  需用一定的控制结构；需采用堆栈技术

- 不可撤回

  需要使用规划技术

问题全域可预测否

有些问题它的全域可预测，如水壶问题、定理证明这些问题结局肯定，可只用开环控制结构。

有些问题的全域不可预测，如变化环境下机器人的控制，特别是危险环境下工作的机器人随时可能出意外，必须利用反馈信息，应使用闭环控制结构。

问题要求的是最优解还是较满意解

一般说来，最佳路径问题的计算较任意路径问题的计算要困难。如果使用的启发式方法不理想，那么对这个解的搜索就不可能很顺利。有些问题要求找出真正的最佳路径，可能任何启发式法都不能适用。因此，得进行耗尽式搜索。