

Chapter 3

Structures and Strategies for State Space Search

（状态空间搜索的结构和策略）



搜索策略

- ❖ 人工智能要解决的问题大多数是结构不良或者非结构的问题，对这样的问题一般不存在成熟的求解算法，而只能利用已有的知识一步步地摸索着前进。
- ❖ 在这个过程中，存在着如何寻找一条推理路线，使得付出的代价尽可能地少，而问题又能够得到解决。我们称寻找这样路线的过程为搜索。



图搜索结构与技术

- 状态图搜索
- 与或树搜索
- 博弈树搜索（第四章）



3.0 Introduction

3.1 Structures for state space search

3.1.1 Graph Theory

3.1.2 The Finite State Machine

3.1.3 The State Representation of Problem

3.2 Strategies for State Space Search

3.2.1 Data-Driven and Goal-Driven Search

3.2.2 Implementing Graph Search

3.2.3 Depth-First and Breadth-First Search

1. Breadth-First Search

2. Depth-First Search

3.3 Using the state space to represent reasoning with the propositional and predicate calculus

3.3.1 State space description of a logic system

3.3.2 And / Or Graphs

3.3.3 Further Examples and Applications

Exercises P.121



3.0 Introduction

- Is **the problem solver** guaranteed (保证) to find a solution?
- Will **the problem solver** always terminate? Can it become **caught in** (陷入) an infinite loop (死循环) ?
- When a solution is found ,is it **guaranteed** to be optimal (最优) ?
- What is the complexity (复杂度) of **the search process** in terms of **time usage** (耗费) ?
Memory usage?



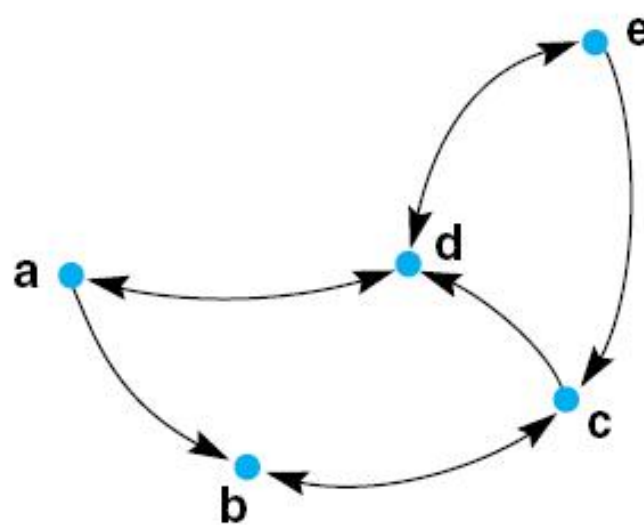
- By **representing a problem as a state graph**, we can use the theory of state space search to answer these questions.
- Euler invented **the graph theory** to solve the “bridges of konigsberg problem” (哥尼斯堡七桥问题)
- This **problem** is “**If there is a walk around the city that crosses each bridge exactly once**”.



3.1 Structures for State Space Search

3.1.1 Graph Theory

- A labeled (带标签的, 如a, b) directed graph.

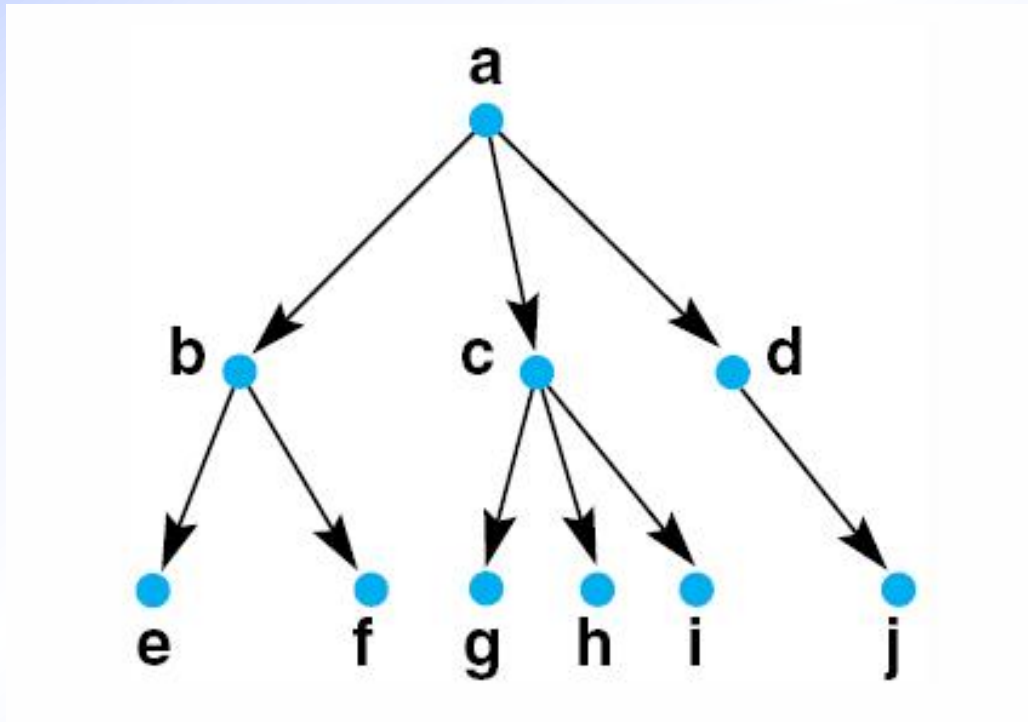


Nodes = {a,b,c,d,e}

Arcs = {(a,b),(a,d),(b,c),(c,b),(c,d),(d,a),(d,e),(e,c),(e,d)}



- **A rooted tree**
(parent, child, siblings, path, connect)

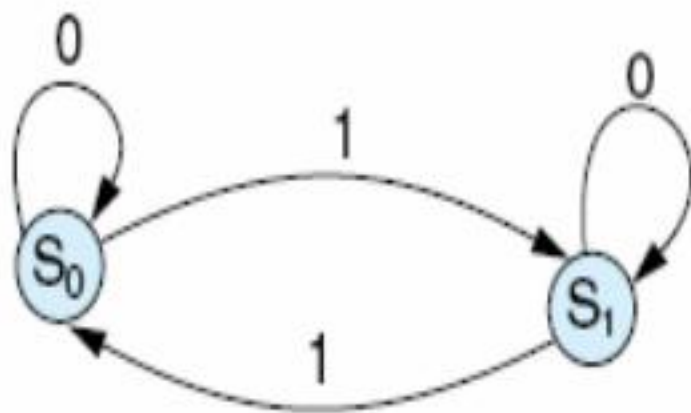


3.1.2 The Finite State Machine

- Definition

- a **finite state machine** (有限状态自动机) is an **ordered triple** (S, I, F) , where:
- **S is a finite set of states in a connected graph $s_1, s_2, s_3 \dots s_n$**
- **I is a finite set of input values $i_1, i_2 \dots i_m$**
- **F is a state transition function** that for any $i \in I$, describes its effect on the states S of the machine, thus $i \in I, F_i : (S \rightarrow S)$.
- If the machine is in state **s_j** and input **i** occurs, the next state of the machine will be $F_i(s_j)$





(a)

	0	1
S_0	S_0	S_1
S_1	S_1	S_0

(b)



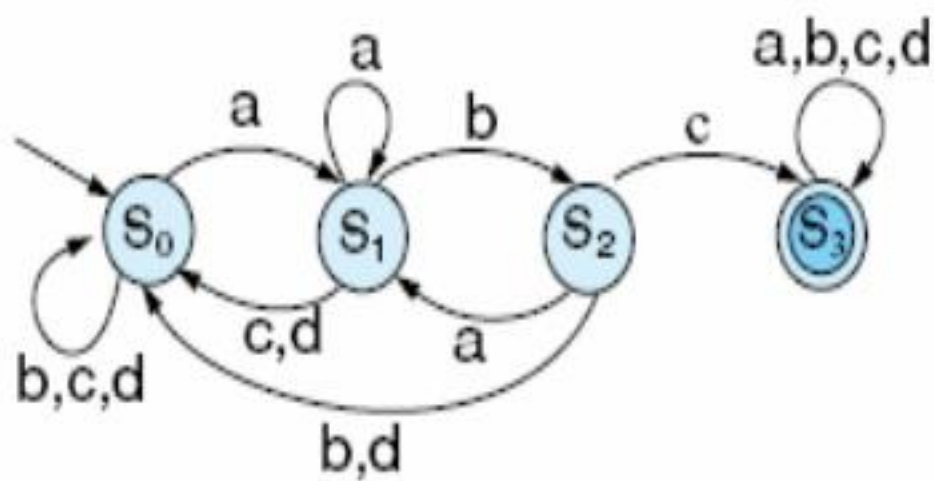
- Definition:

finite state acceptor (moore machine)

有限状态接收器 (摩尔机)

- A **finite state acceptor** is a finite state machine (S, I, F) , where:
- $\exists s_0 \in S$ such that **the input stream starts at s_0**
- $\exists s_n \in S$, an accept state. **The input stream is accepted in that state** (if it terminates).
- The finite state acceptor is represented as $(S, s_0, \{ s_n \}, I, F)$





(a)

➔

	a	b	c	d
S_0	S_1	S_0	S_0	S_0
S_1	S_1	S_2	S_0	S_0
S_2	S_1	S_0	S_3	S_0
S_3	S_3	S_3	S_3	S_3

(b)



3.1.3 The State Representation of Problem

- Definition

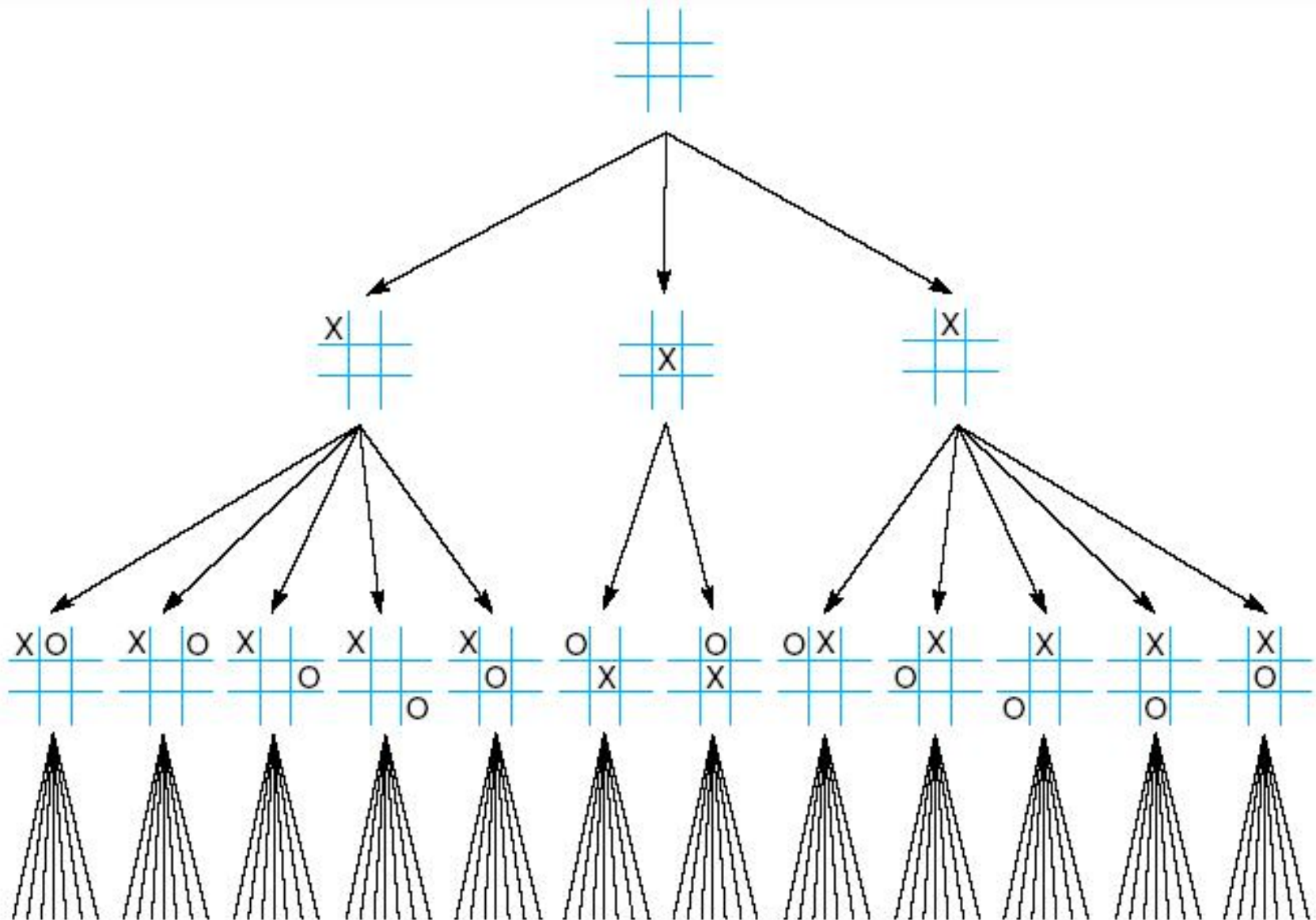
- A state space is represented by a four-tuple $[N, A, S, GD]$, where:
- N is the set of nodes (states) of the graph.
- A is the set of arcs between nodes.
- S is a nonempty subset of N , contains the start state of the problem.
- GD is a nonempty subset of N , contains the goal state of the problem.



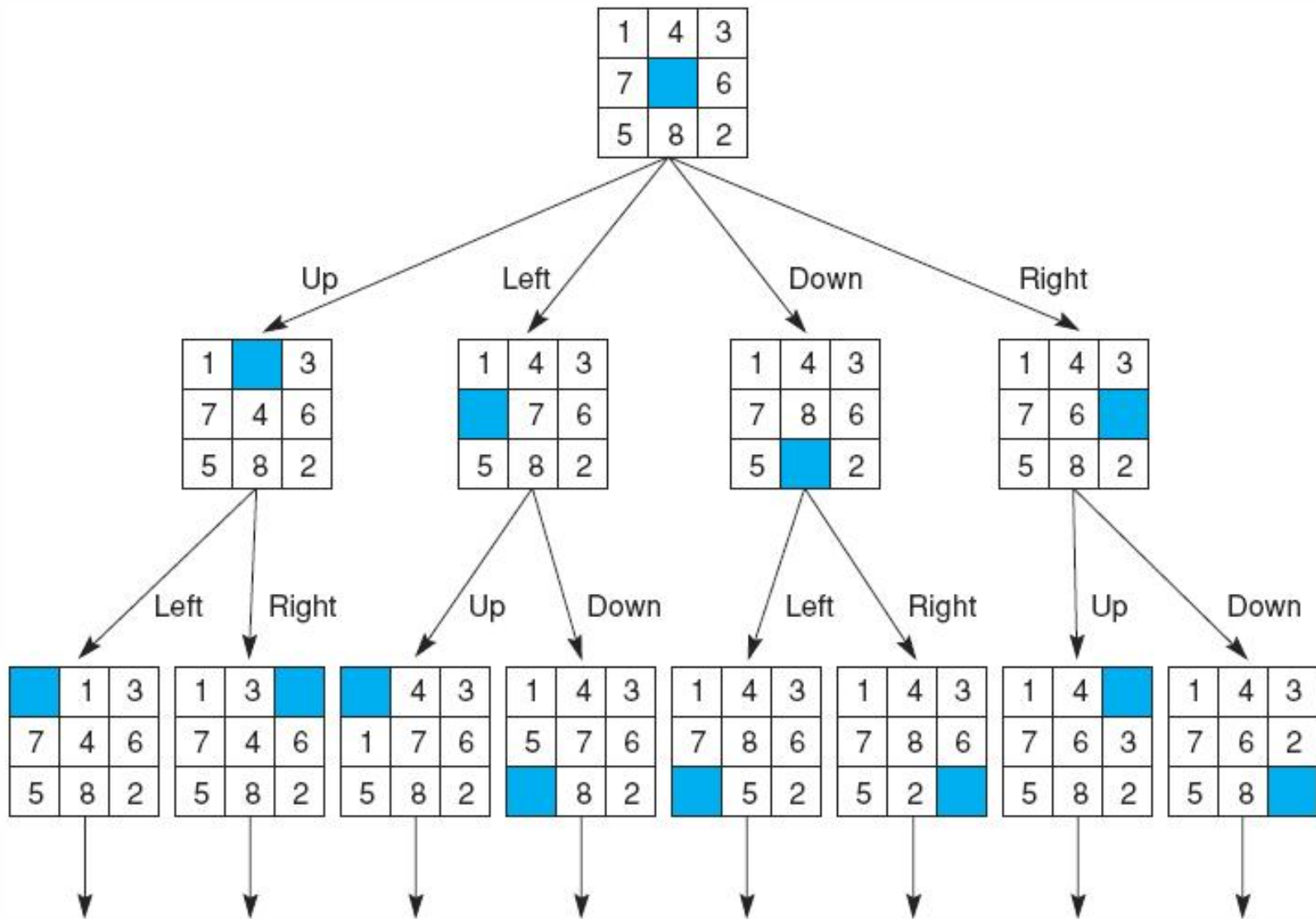
- **The states in GD** are described using either:
 1. A measurable property (可测量属性) of **the states** encountered in the search.
 2. A measurable property (可测量属性) of **the path** developed in the search, for example, the sum of the transition costs (转换代价) for the arcs of the path.
- **A solution path** is a path through this graph from a node in **S** to a node in **GD**.



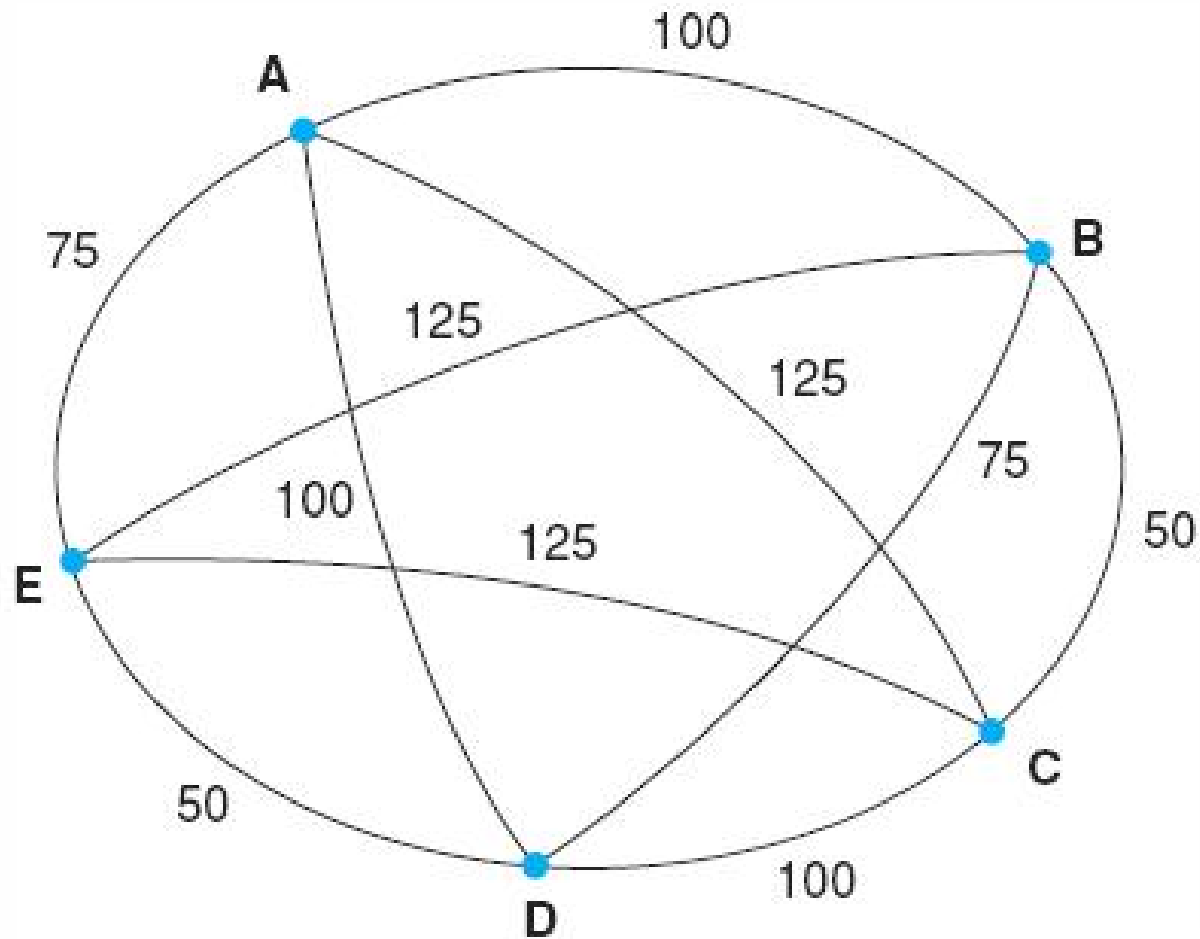
Example 3.1.1 TIC-TAC-TOE (一字棋)

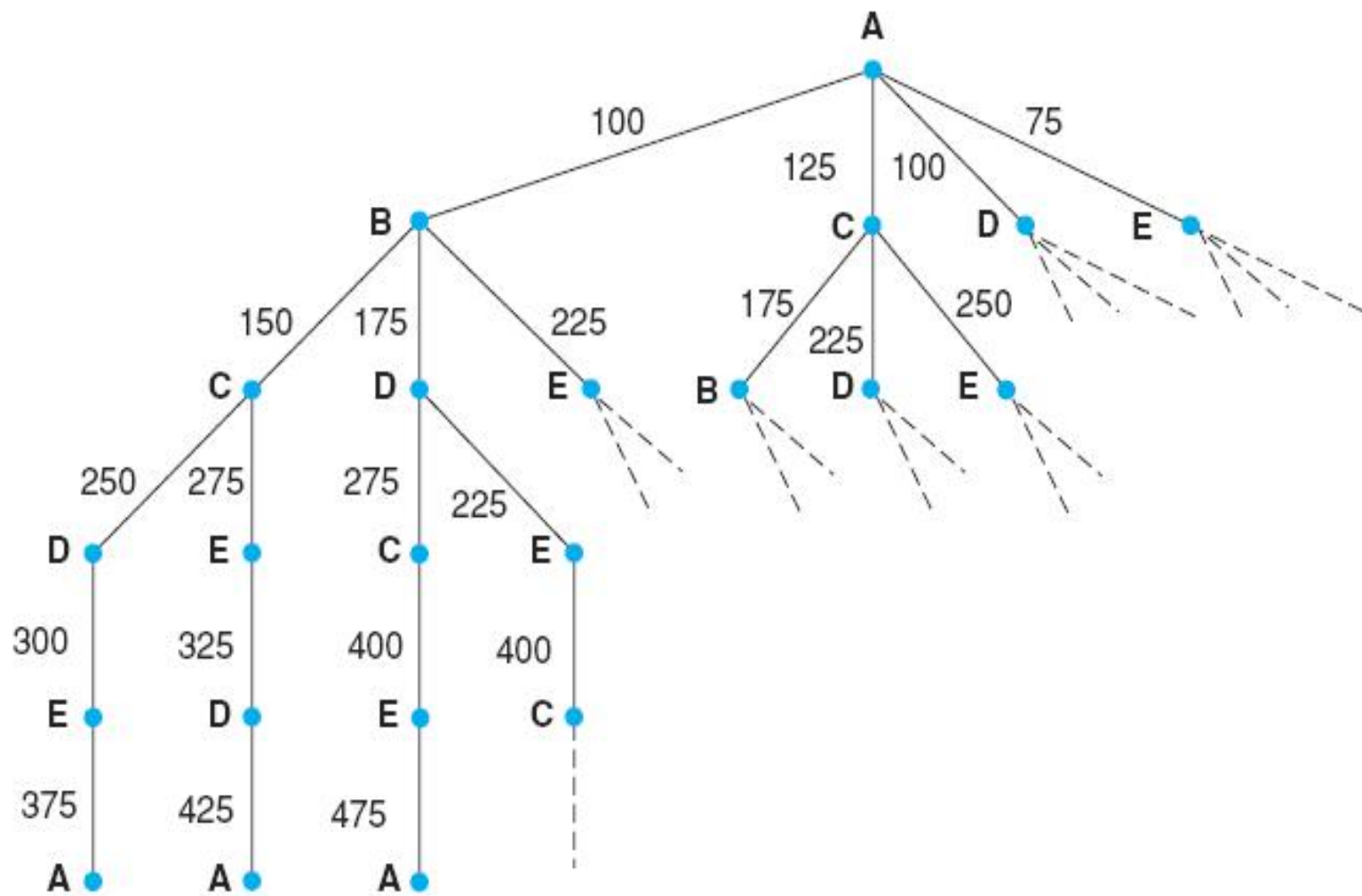


Example 3.1.2 THE 8-PUZZLE



Example 3.1.3 The **T**raveling **S**alesperson **P**roblem (**TSP**)





- The complexity of **exhaustive search** (穷举搜索) in **the traveling salesperson problem** is $(N-1)!$
- **Several techniques** can **reduce** this search complexity.
- **One** is called **branch and bound** (分支限界) .
- **Another strategy** for controlling search is called **nearest neighbor** (最近邻) .

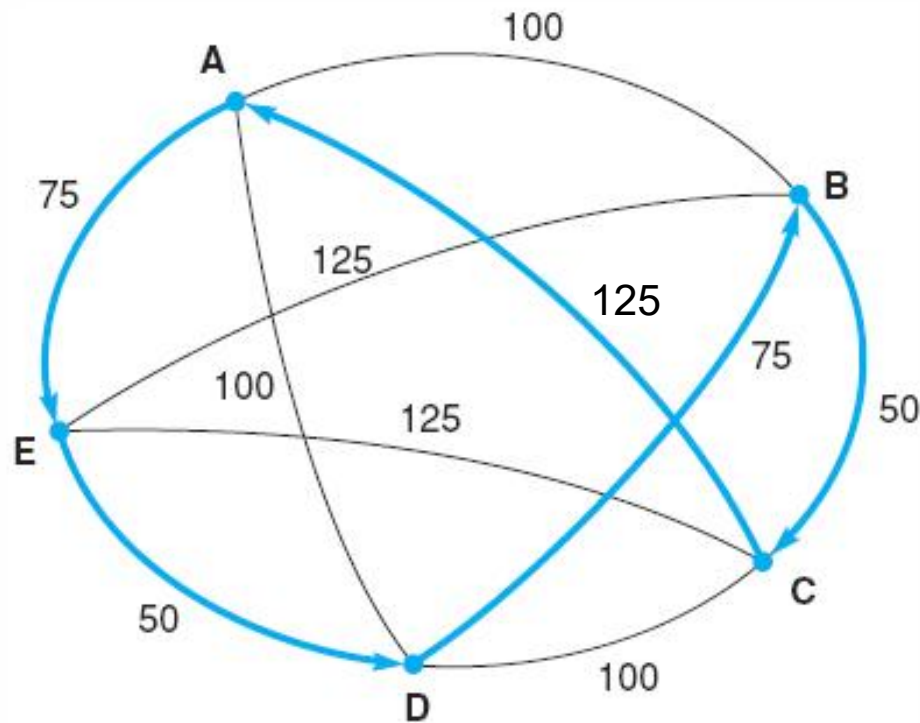


The **nearest neighbor** algorithm
---- a kind of **greedy** algorithm

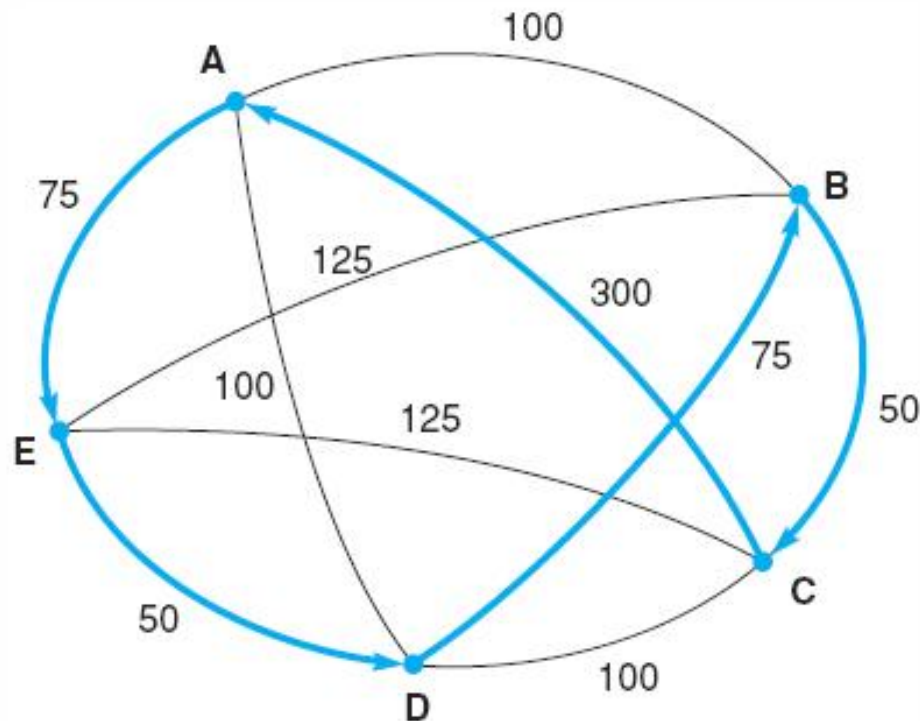
- The **searching strategy** is :
- “**Go to the closest un-visited city.**”



- An instance of **the travelling salesperson problem** with **the nearest neighbor path** in bold (Figure 3.9, not 3.11)
- Note this **path** (A, E, D, B, C, A), at a **cost** of **375**



- Another instance of the TSP with the nearest neighbor path in bold (粗线).
- Note this path (A, E, D, B, C, A), at a cost of 550, is not the shortest path.
- The high cost of arc (C, A) defeated the heuristics



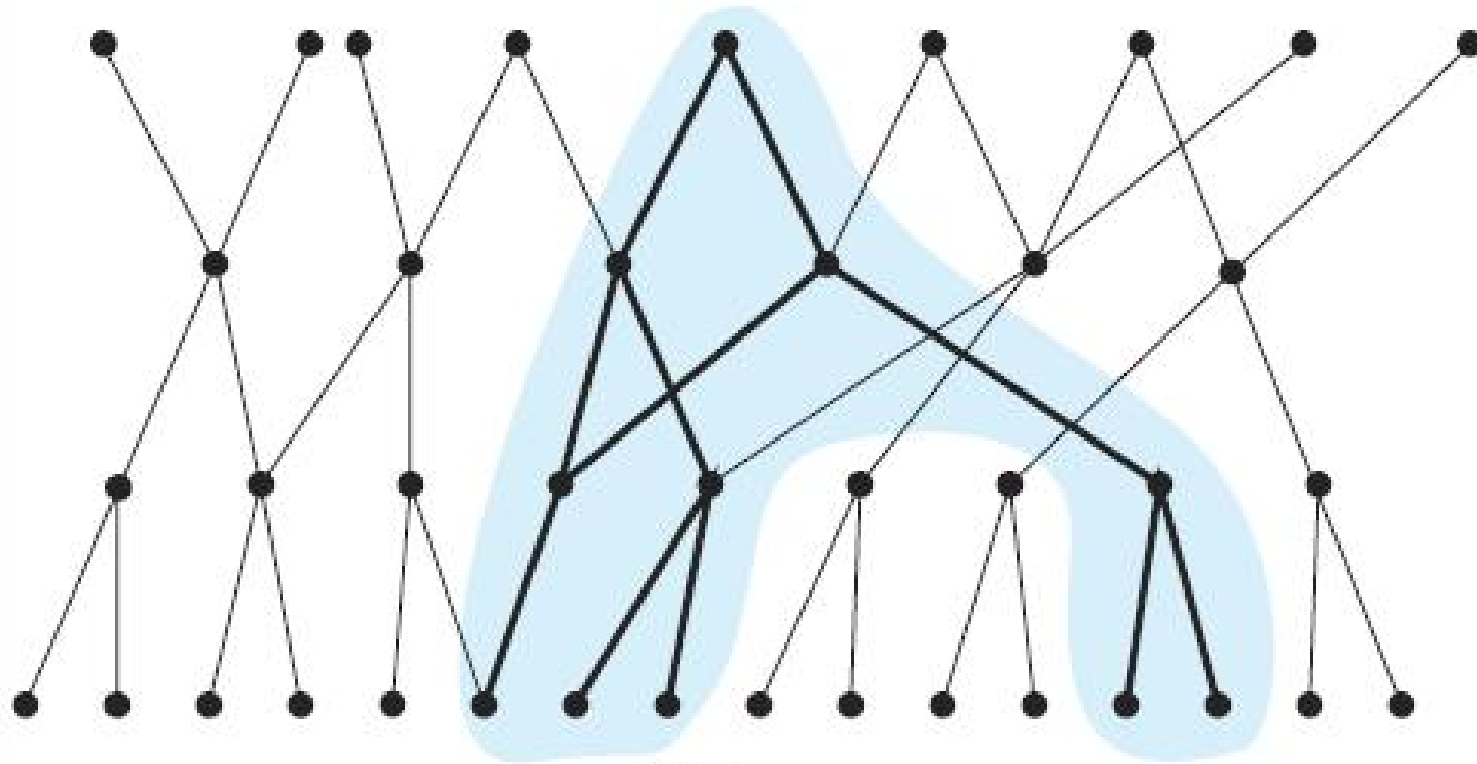
3.2 Strategies for State Space Search

3.2.1 Data-Driven and Goal-Driven Search

- In **data-driven** search, sometimes called **forward chaining** (前向链), the problem solver begins with **the given facts** of the problem and **a set of legal moves** (走步) or **rules** for changing state.
- Search proceeds by **applying rules to facts** to produce new facts, **which are in turn used** by the rules to generate more new facts.
- This process **continue until** it generates a path that satisfies the goal condition.



Goal



Data

Direction of reasoning
↑



Cases for **Data-Driven** method

1. **Most of the data are given** in the initial problem statement.
2. There are **a large number of potential goals**.
3. It is difficult **to form a goal or hypothesis**.



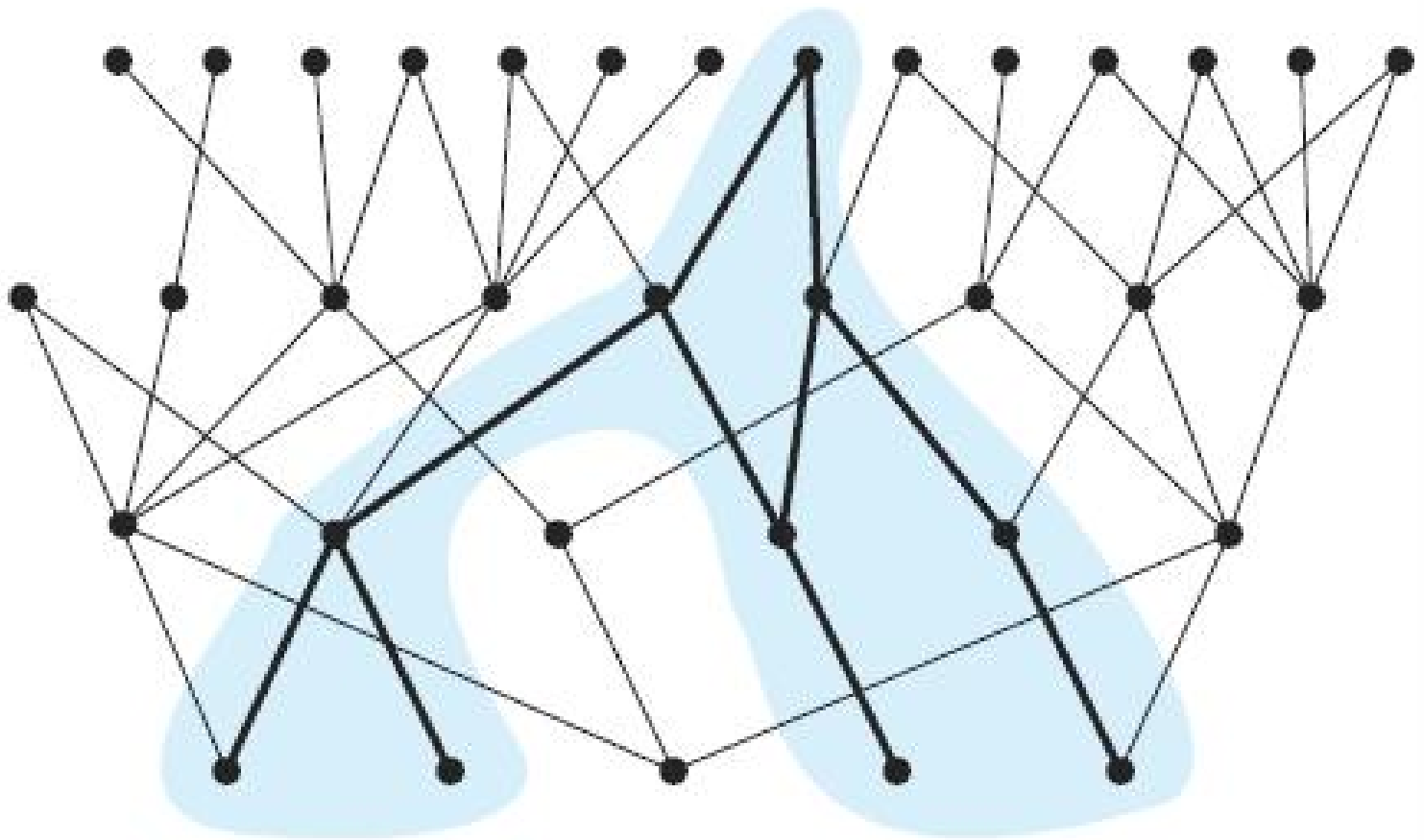
Goal-Driven search

- take **the goal** that we want to solve
- See what **rules** or **legal moves** could be used **to generate this goal** and determine what **conditions** (前提条件) must be **true** to use them.
- These **conditions** become **the new goals**, or **subgoals** (子目标) for the search.
- Search continue, **working backward** (逆向) through **successive subgoals** until it **works back to** the facts of the problem.



Direction of
reasoning

Goal



Data

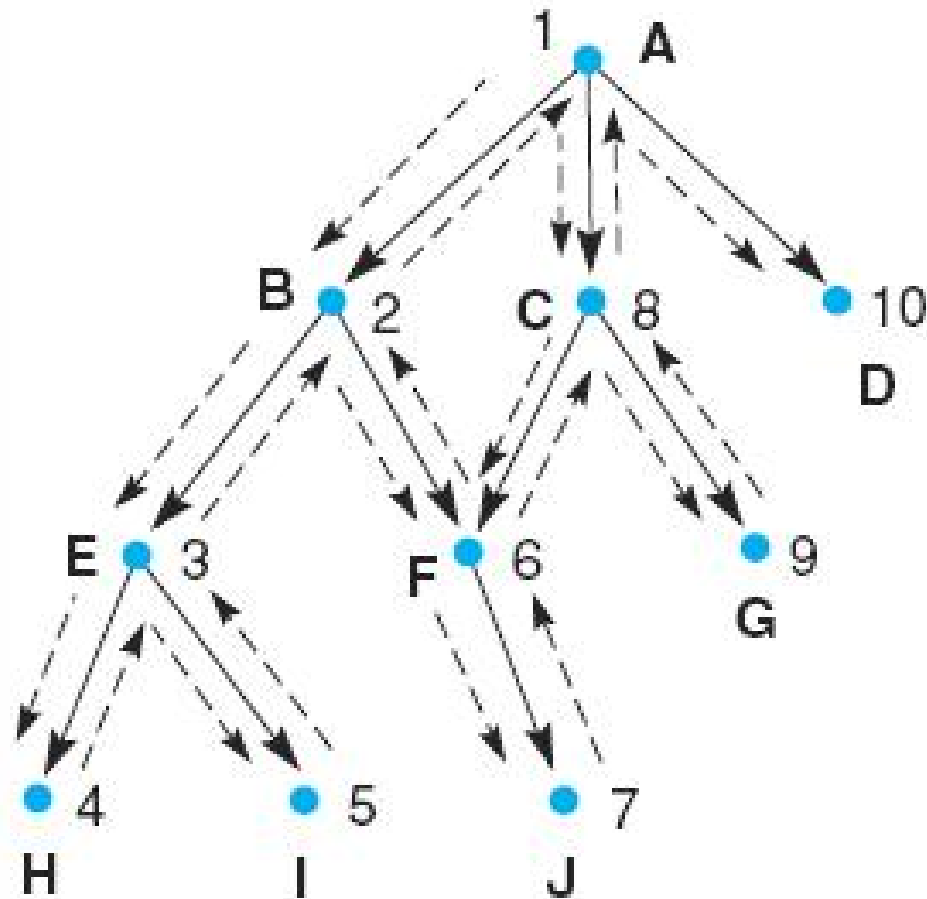


Cases for **Goal-Driven** method

1. A **goal** or **hypothesis** (假设) is **given** in the problem statement or can easily be formulated.
2. There are **a large number of rules** that **match the facts** and thus produce **an increasing number of irrelevant conclusions**.
3. Problem **data are not given** but must be **acquired** by the problem solver.



3.2.2 Implementing Graph Search Backtracking :



Backtracking

- **SL** : for **S**tate **L**ist of
the **C**urrent **P**ath (**CP**)
- **NSL** : for **N**ew **S**tate **L**ist
- **DE** : for **D**ead **E**nds (死端, 死胡同)
- **CS** : for **C**urrent **S**tate



Function backtrack;

begin

SL:=[Start]; NSL:=[Start]; DE:=[]; CS:=start;

while NSL≠ [] do

begin

if CS = goal then return SL;

if CS has no children

then // Backtrack

begin

while SL is not empty and CS == first (SL) do

begin

add CS to DE;

remove first element from SL;

remove first element from NSL;

CS := first element of NSL; // sibling(兄弟) ?

end

add CS to SL;

end

else // Forward

begin

place children of CS on NSL; // Front, Stack

CS := first element of NSL; // First child

add CS to SL;

end

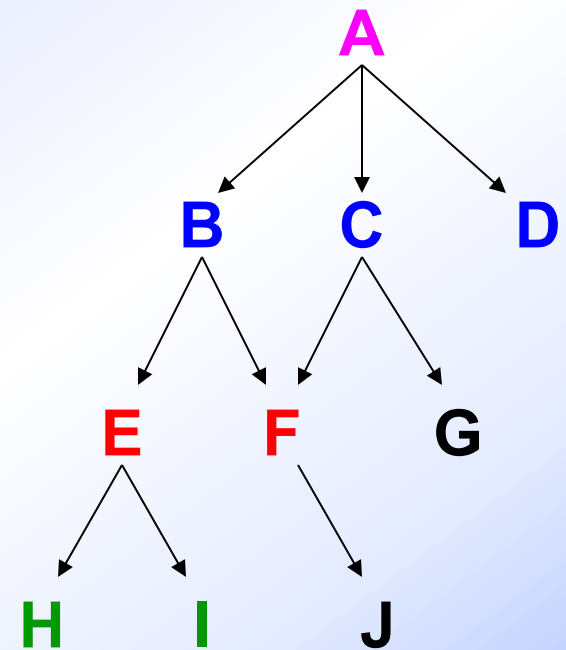
end; // while

return FAIL;

end.



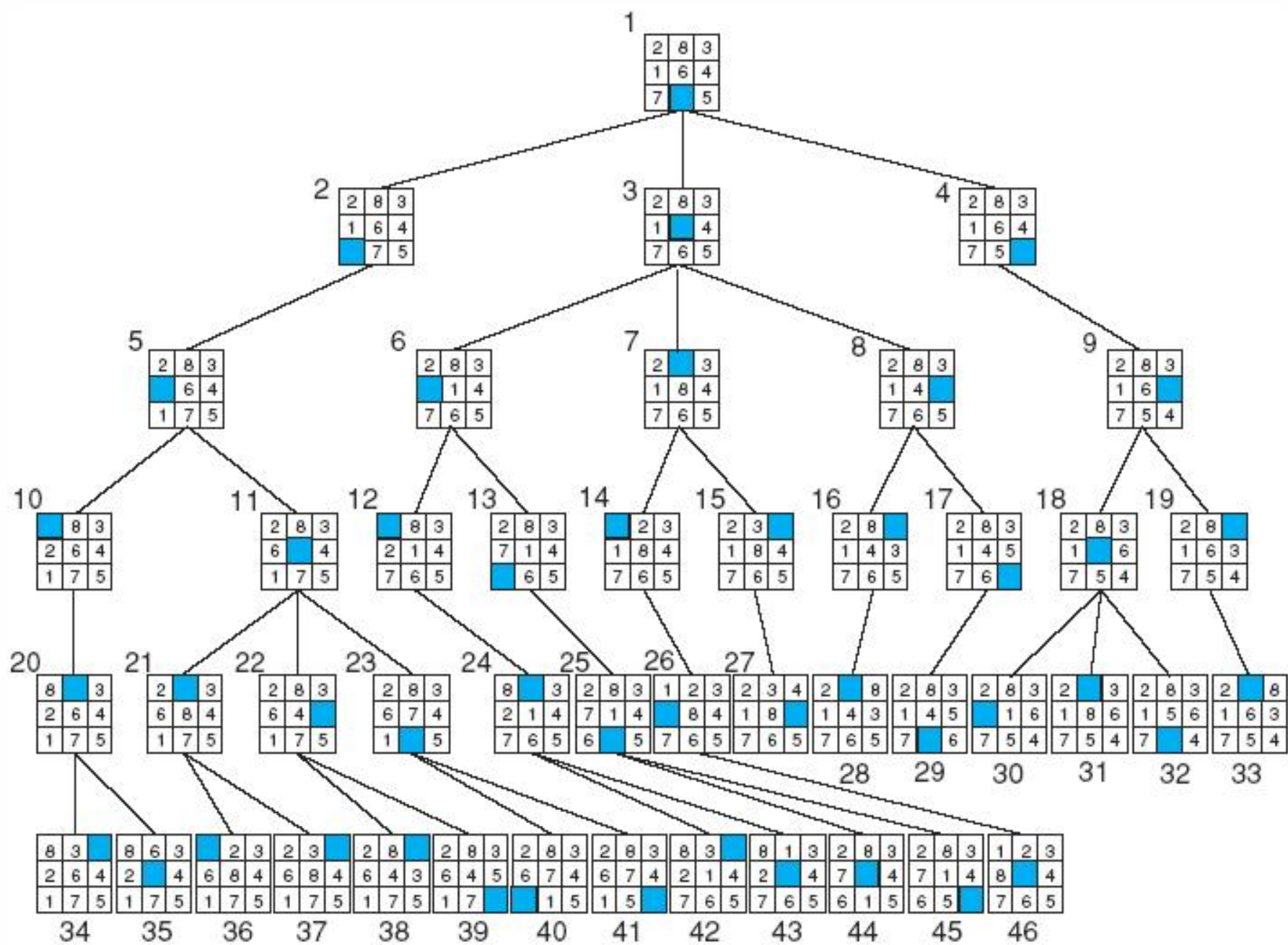
	CS	SL	NSL	DE
0	A	[A]	[A]	[]
1	B	[BA]	[BCDA]	[]
2	E	[EBA]	[EFBCDA]	[]
3	H	[HEBA]	[HIEFBCDA]	[]
4	I	[IEBA]	[IEFBCDA]	[H]
4'	E	[EBA]	[EFBCDA]	[IH]
4''	F	[BA]	[FBCDA]	[EIH]
5	F	[FBA]	[FBCDA]	[EIH]
6	J	[JFBA]	[JFBCDA]	[EIH]
7	C	[CA]	[CDA]	[BFJEIH]
8	G	[GCA]	[GCDA]	[BFJEIH]



3.2.3 Depth-First and Breadth-First Search

1. Breadth-First Search





Goal

第零层

2	8	3
1		4
7	6	5

S0

第一层

2	8	3
	1	4
7	6	5

B1

2		3
1	8	4
7	6	5

B2

2	8	3
1	4	
7	6	5

B3

2	8	3
1	6	4
7		5

B4

第二层

	8	3
2	1	4
7	6	5

C1

2	8	3
7	1	4
	6	5

C2

	2	3
1	8	4
7	6	5

C3

	2	3
1	8	4
7	6	5

C4

2	8	
1	4	3
7	6	5

C5

2	8	3
1	8	5
7	6	

C6

2	8	3
1	6	4
	7	5

C7

2	8	3
1	6	4
7	5	

C8

第三层

8		3
2	1	4
7	6	5

D1

2	8	3
7	1	4
6		5

D2

1	2	3
	8	4
7	6	5

D3

2	3	4
1	8	
7	6	5

D4

2		8
1	4	3
7	6	5

D5

2	8	3
1	4	5
7		6

D6

2	8	3
	6	4
1	7	5

D7

2	8	3
1	6	
7	5	4

D8

第四层

8	3	
2	1	4
7	6	5

E1

8	1	3
2		4
7	6	5

E2

2	8	3
7		4
6	1	5

E3

2	8	3
7	1	4
6	5	

E4

1	2	3
8		4
7	6	5

Sg

1	2	3
7	8	4
	6	5

E6

2	3	4
1		8
7	6	5

E7

找到目标状态，
结束搜索

广度优先搜索



- **open** : records **states** that have been **generated** but **not examined**.

(current “leaf”)

NSL （回溯算法**Backtracking**的新状态列表）

FIFO （队列：先进先出）

Right （插入**Open**表的右端）

- **closed** : records **states** already **examined**.

SL、DE （回溯算法**Backtracking**的状态列表、死端）



```
Function breadth_first_search;  
begin  
    open := [ start ] ;  
    closed := [ ] ;  
    While open ≠ [ ] do  
        begin  
            remove leftmost state from open ,call it X;  
            if X is a goal then return SUCCESS  
            else begin  
                generate children of X;  
                put X on closed;  
                discard children of X if already on open or closed;  
                put remaining children on right end of open  
            end  
        end  
    return FAIL  
end.
```



1 open=[**A**]; closed = []

2 open=[**B,C,D**]; closed = [**A**]

3 open=[**C,D,E,F**]; closed=[**B,A**]

4 open=[**D,E,F,G,H**]; closed=[**C,B,A**]

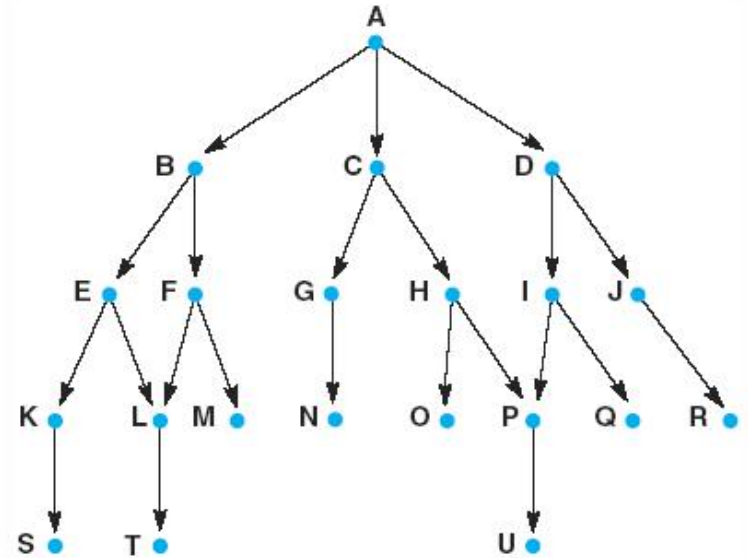
5 open=[**E,F,G,H,I,J**]; closed=[**D,C,B,A**]

6 open=[**F,G,H,I,J,K,L**]; closed=[**E,D,C,B,A**]

7 open=[**G,H,I,J,K,L,M**]; closed=[**G,F,E,D,C,B,A**]

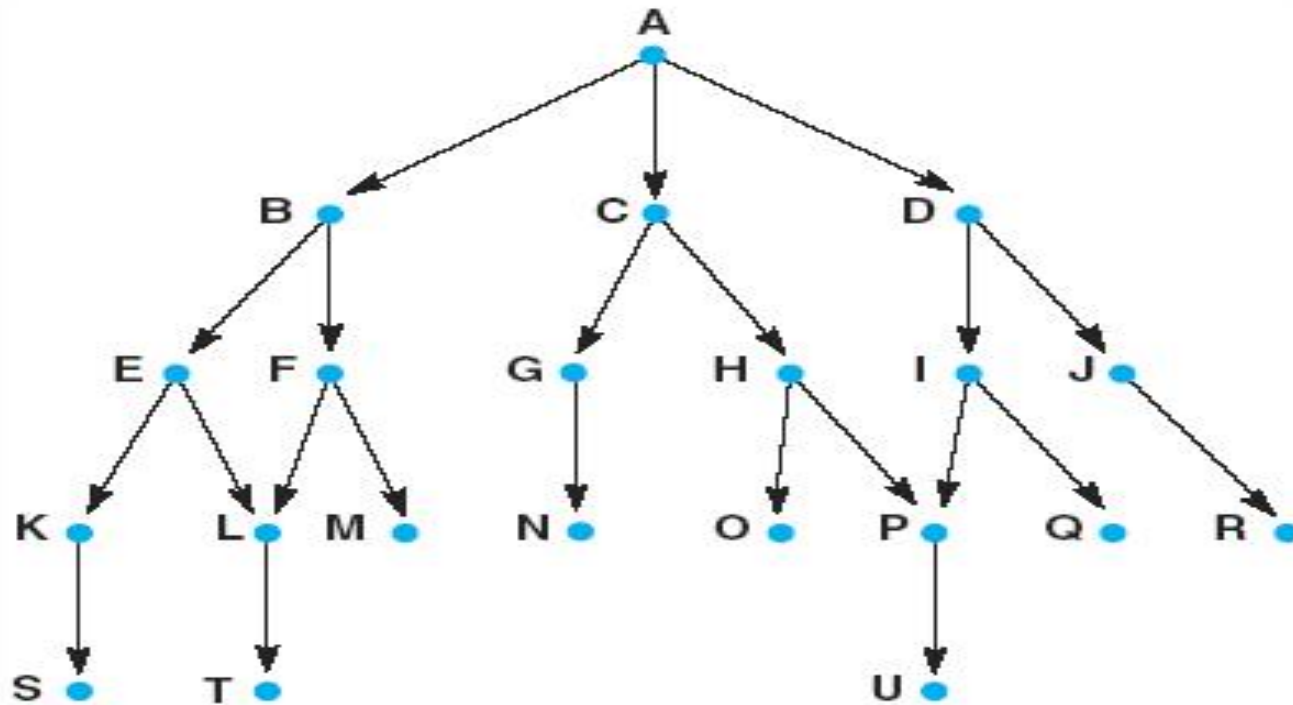
8 open=[**HIJKLMN**]; closed=[**G,F,E,D,C,B,A**]

9 and so on until either **U** is found or open = []



- **A state** may be saved along with **its parent state**
- that is, saved as a (**state, parent**) pair.
- When **the goal state** is found, **the solution path** can be easily **constructed**.





- 1 open=[(A, nil)]; closed=[]
- 2 open=[(B, A), (C, A), (D, A)]; closed=[(A, nil)]
- 3 open=[(C, A), (D, A) , (E, B), (F, B)]; closed =[(B, A), (A, nil)]

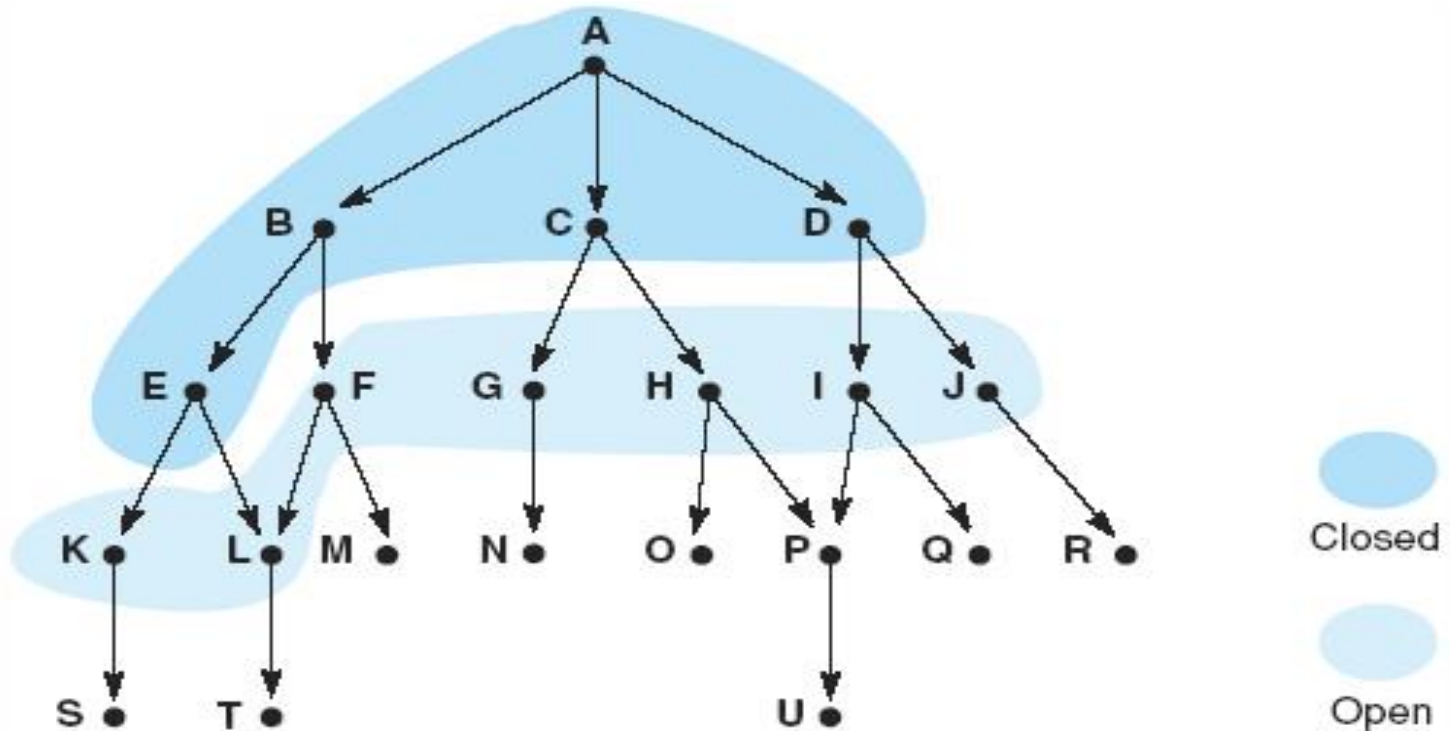


At **iteration 6** of **breadth-first** search.

States on **open** and **closed** are highlighted.

open = [(F,B), (G,C), (H,C), (I,D), (J,D), (K,E), (L, E)]

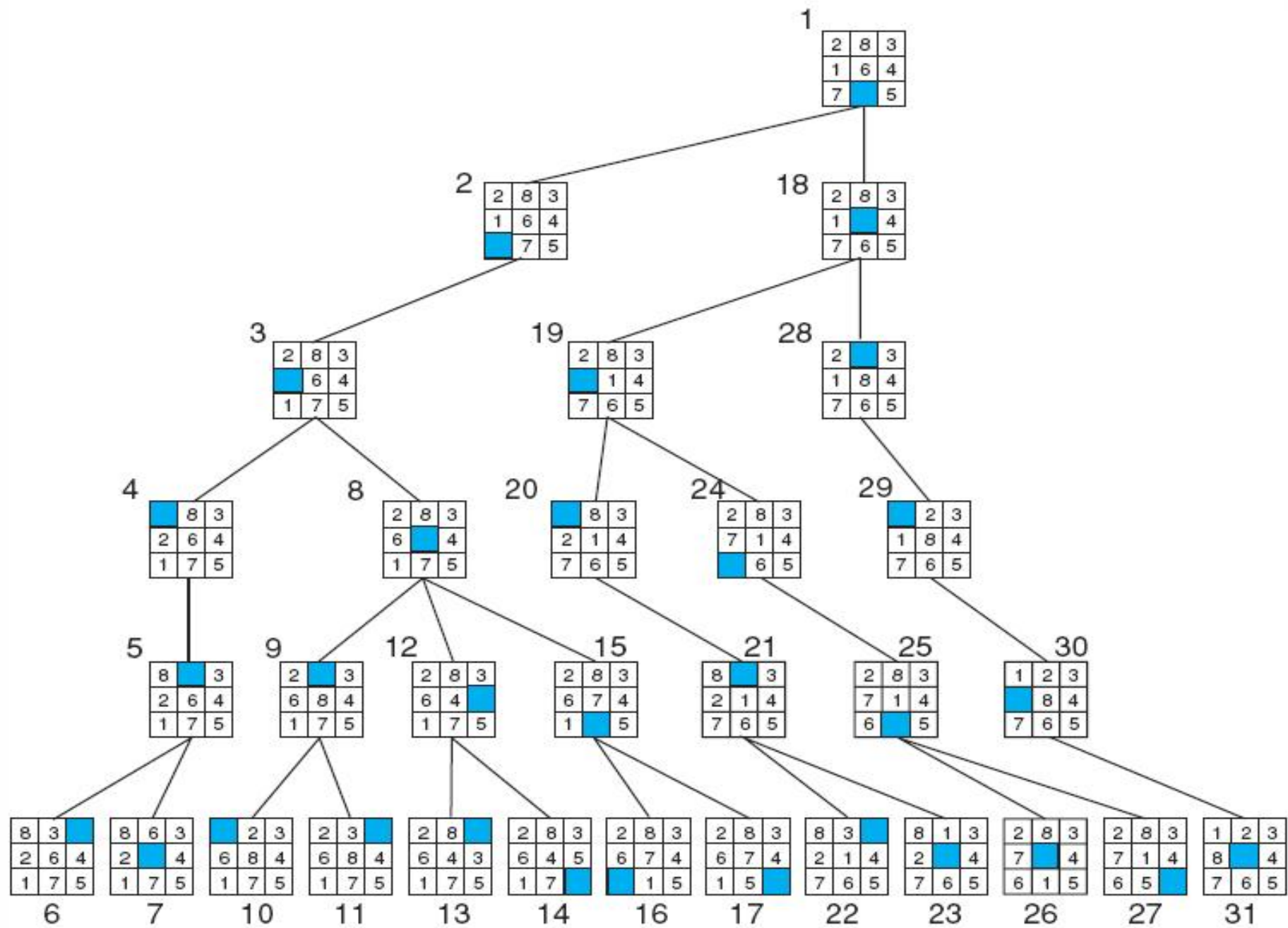
closed = [(E,B), (D,A), (C,A), (B,A), (A, nil)]



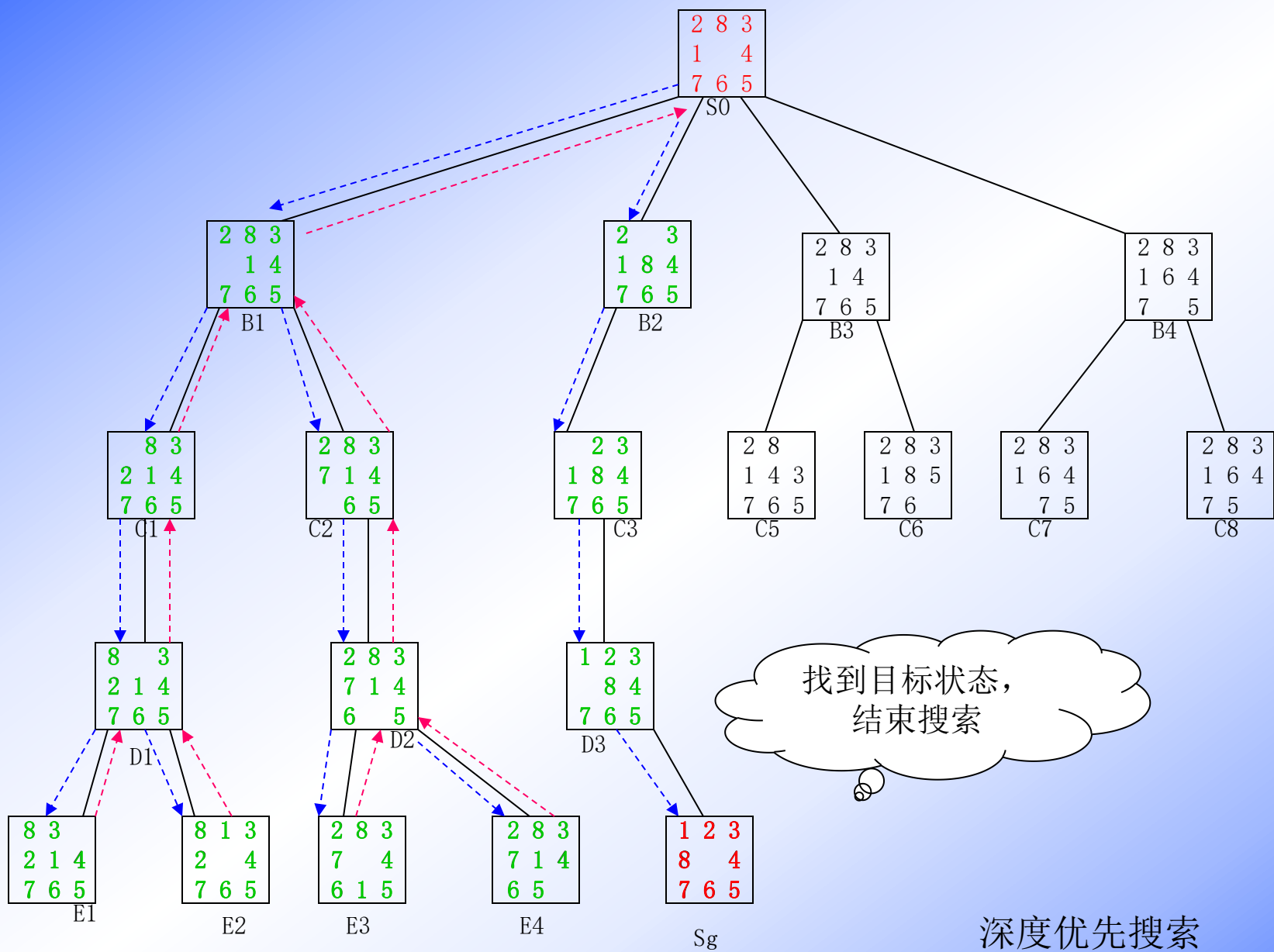
3.2.3 Depth-First and Breadth-First Search

2. Depth-First Search





Goal



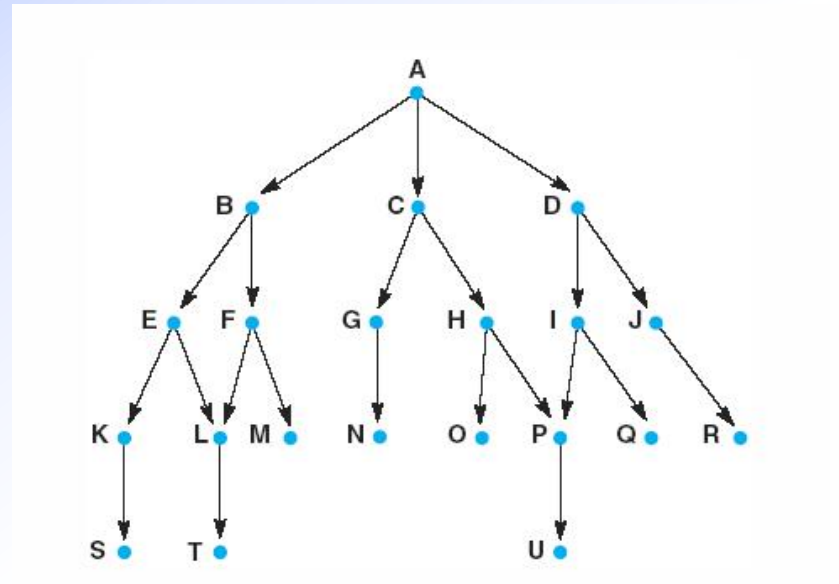
深度优先搜索



```
Function depth_first_search;  
begin  
  open := [ start ] ;  
  closed := [ ] ;  
  While open ≠ [ ] do  
    begin  
      remove leftmost state from open ,call it X;  
      if X is a goal then return SUCCESS  
      else begin  
        generate children of X;  
        put X on closed;  
        discard children of X if already on open or closed;  
        put remaining children on left end of open  
      end  
    end  
  end  
  return FAIL  
end.
```



- 1 open=[**A**]; closed=[]
- 2 open=[**B,C,D**]; closed=[**A**]
- 3 open=[**E,F,C,D**]; closed=[**B,A**]
- 4 open=[**K,L,F,C,D**]; closed=[**E,B,A**]
- 5 open=[**S,L,F,C,D**]; closed=[**K,E,B,A**]
- 6 open=[**L,F,C,D**]; closed=[**S,K,E,B,A**]
- 7 open=[**T,F,C,D**]; closed=[**L,S,K,E,B,A**]
- 8 open=[**F,C,D**]; closed=[**T,L,S,K,E,B,A**]
- 9 open=[**M,C,D**]; closed=[**F,T,L,S,K,E,B,A**]
- 10 open=[**C,D**]; closed=[**M,F,T,L,S,K,E,B,A**]
- 11 open=[**G,H,D**]; closed=[**C,M,F,T,L,S,K,E,B,A**]
- 12 and so on **until** either U is found **or** open=[]



有界深度优先搜索

有界深度优先搜索的搜索过程为：

- 1) 把初始节点 S_0 放入OPEN表中，置 S_0 的深度 $d(S_0)=0$ 。
- 2) 如果OPEN表为空，则问题无解，退出。
- 3) 把OPEN表的第一个节点（记为节点n）取出放入CLOSED表。



- 4) 考察节点n是否为目标节点。若是，则求得了问题的解，退出。
- 5) 如果节点n的深度 $d(\text{节点}n) = S_m$ ，则转第2)步。
- 6) 若节点n不可扩展，则转第2)步。
- 7) 扩展节点n，将其子节点放入OPEN表的首部，并为其配置指向父节点的指针，然后转第2)步。

思考：此算法如何改进，以找到解或最优解？



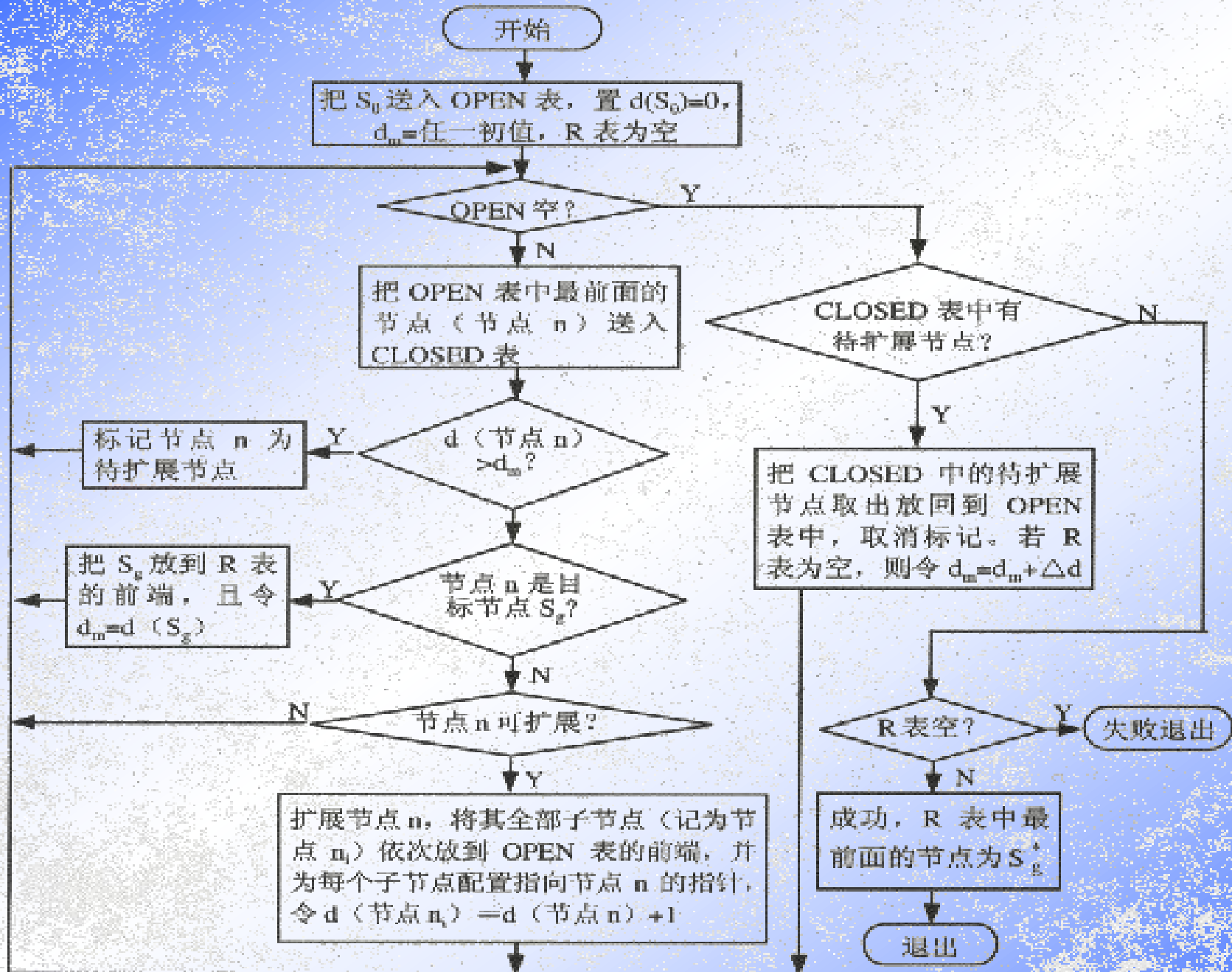


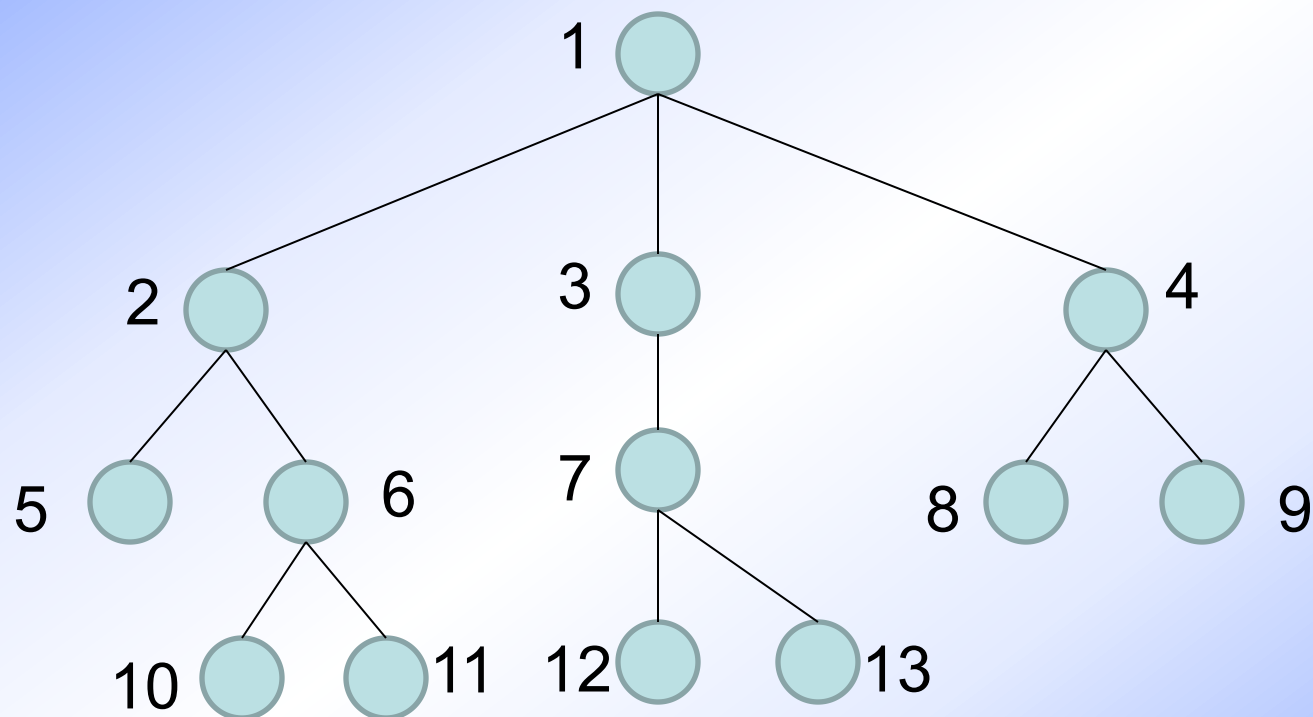
图 4.11 求最优解的有界深度优先搜索流程示意图

3.2.4 Depth-First Search with Iterative Deepening

- ***Depth-first Iterative Deepening*** (迭代加深的深度优先) performs a depth-first search of the space with a ***depth bound of 1***. If it fails to find a goal, it performs another depth-first search with a ***depth bound of 2***. This continues, increasing the ***depth bound by one each time***. At each iteration, the algorithm performs a complete depth-first search to the current depth bound. No information about the state space is retained between iterations.



深度优先迭代加深搜索（每个迭代深度加1）



节点访问顺序：1, 1, 2, 3, 4, 1, 2, 5, 6, 3, 7, 4, 8, 9,
1, 2, 5, 6, 10, 11, 3, 7, 12, 13, 4, 8, 9



3.3 using the state space to represent reasoning with the propositional and predicate calculus

3.3.1 State space description of a logic system

$q \rightarrow p$

$r \rightarrow p$

$v \rightarrow q$

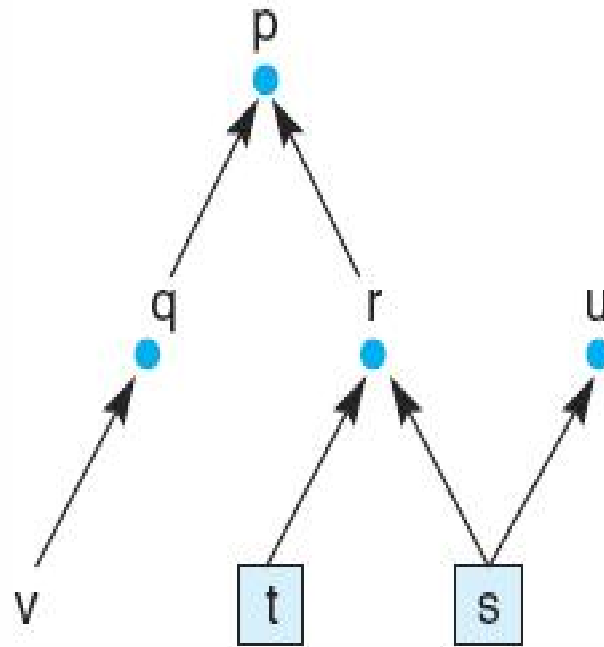
$s \rightarrow r$

$t \rightarrow r$

$s \rightarrow u$

s

t



3.3.2 And / Or Graphs

- Definition : hypergraph (超图)

A hypergraph consists of:

- **N**, a set of **nodes**.
- **H**, a set of **hyperarcs** (超弧) defined by **ordered pairs** in which **the first element** of the pair is **a single node** from **N** and **the second element** is a **subset** of **N**.

- Nodes **in the subset** are “**and nodes**”
- Hyperarcs are also known as **K-connectors**



Fig 3.21 And / or graph of the expression :

$$q \text{ \textcolor{red}{\wedge} } r \rightarrow p.$$

Hyperarcs : $\langle p, \{ q, r \} \rangle$

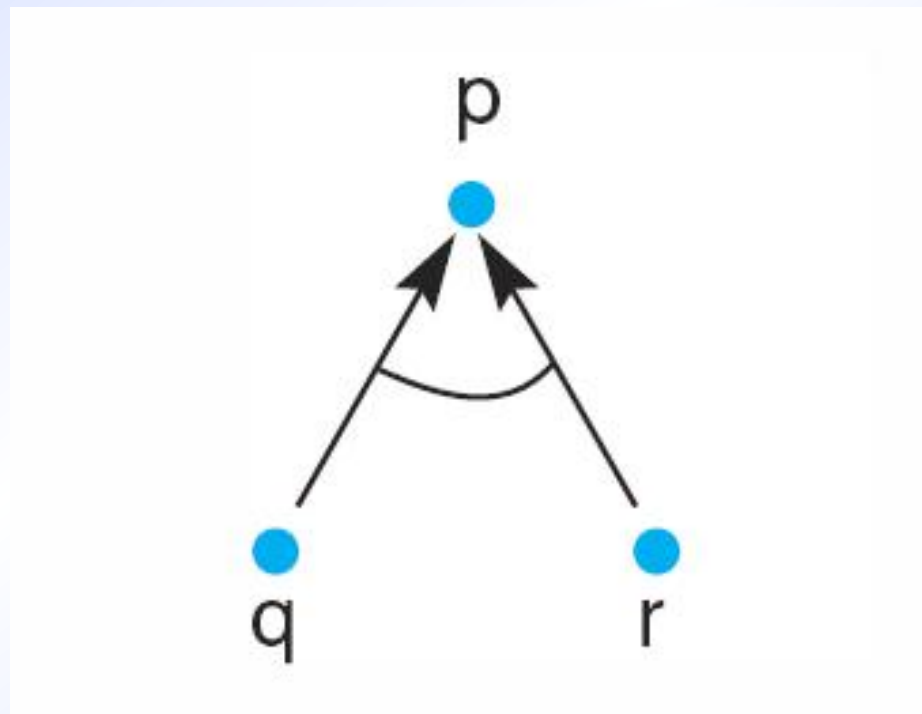
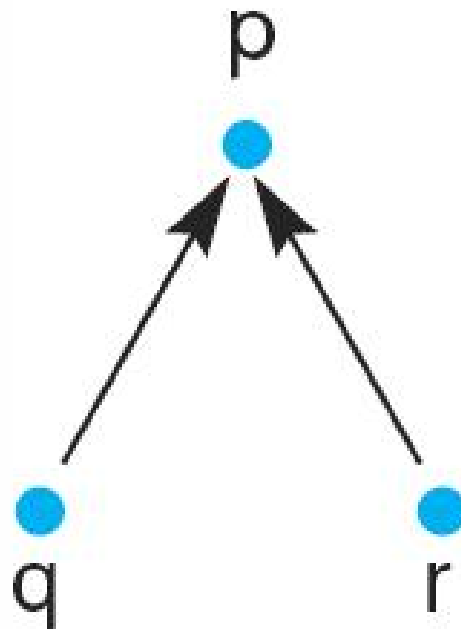


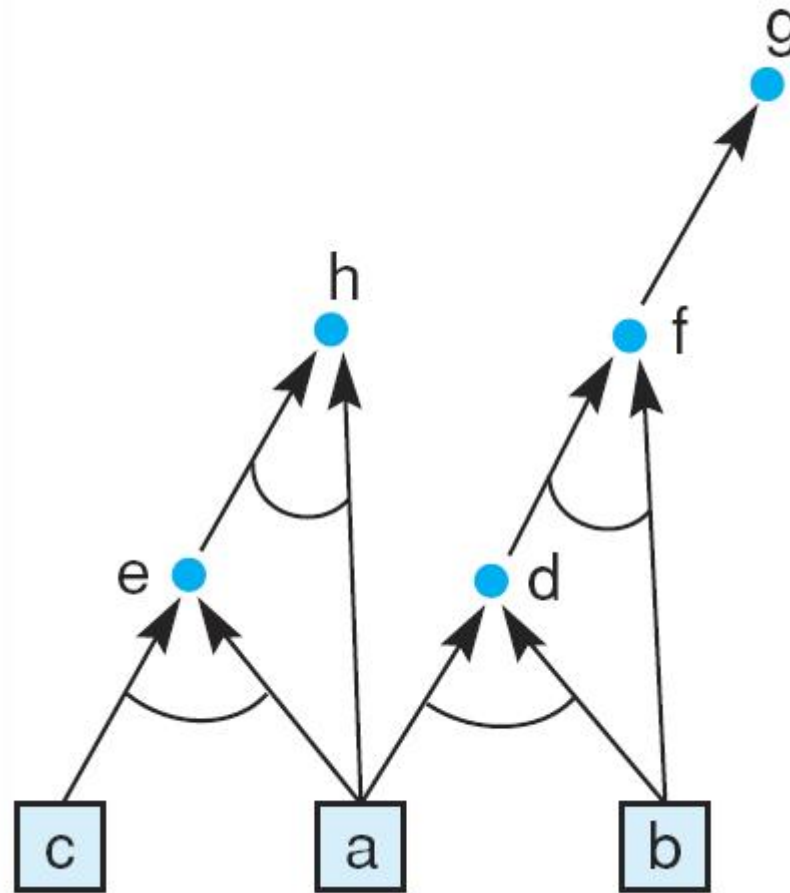
Fig 3.22 And / or graph of the expression :

$$q \vee r \rightarrow p$$

$$K = 1$$



- $a, b, c,$
- $a \wedge b \rightarrow d,$
- $a \wedge c \rightarrow e,$
- $b \wedge d \rightarrow f,$
- $f \rightarrow g,$
- $a \wedge e \rightarrow h$



与/或树表示法

与/或树是用于表示问题及其求解过程的另一种形式化方法，也称为问题归约方法。它把初始问题通过一系列变换最终变为一个子问题集合。



1. 分解

例如，把问题 P 分解为三个子问题 P_1 、 P_2 、 P_3 ，可用图4.2表示。

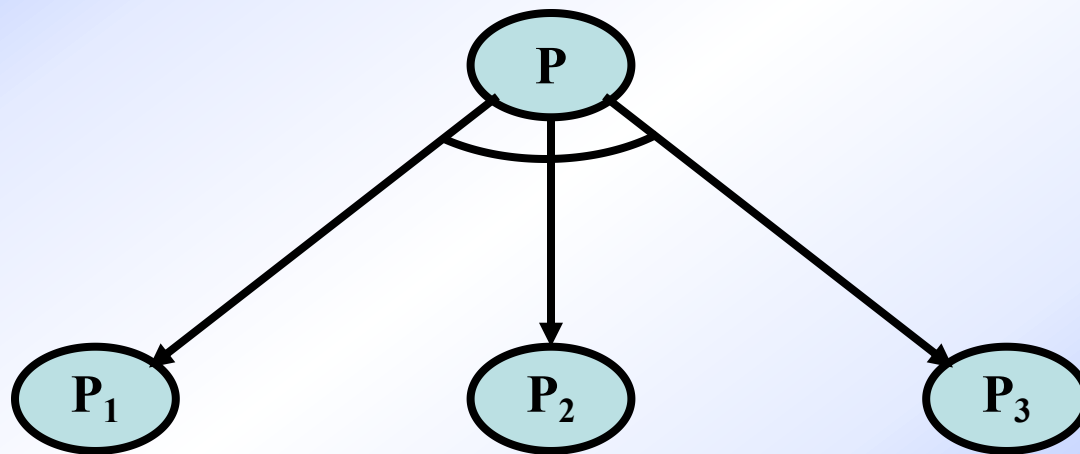


图1 “与”树



2. 等价变换

问题的等价变换过程也可以用一个图表示出来，称为“或”树。

例如，问题 P 被等价变换为新问题 P_1 、 P_2 、 P_3 ，可用图2表示。

上述两种方法也可以结合起来使用，此时的图称为“与/或”树。其中既有“与”节点，也有“或”节点，如图3所示。



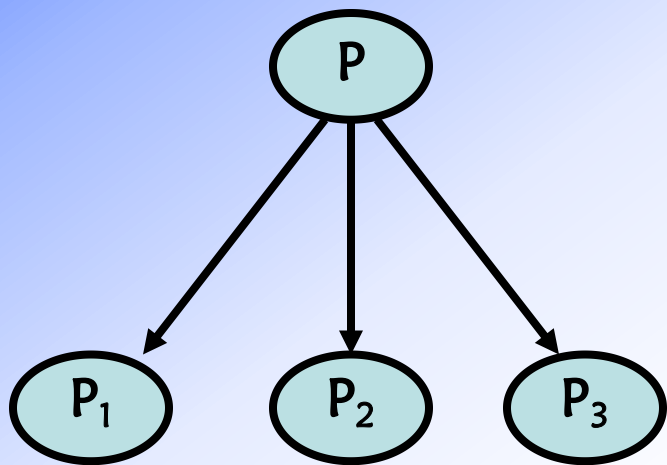


图2 “或” 树

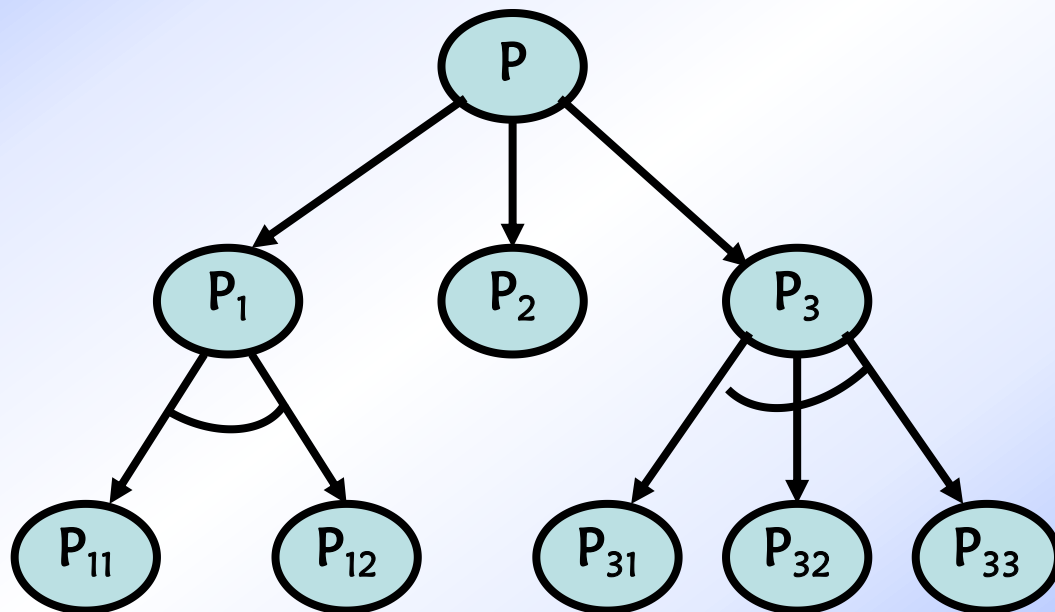


图3 “与/或” 树



3. 本原问题

不能再分解或变换，而且直接可解的子问题称为本原问题

4. 端节点与终止节点

- 端节点概念:

在与/或树中，没有子节点的节点称为端节点;

- 终止节点概念:

本原问题所对应的节点称为终止节点。

- 显然，终止节点一定是端节点，但端节点不一定是终止节点。



5. 可解节点

在与/或树中，满足下列条件之一者，称为可解节点。

- 1) 它是一个终止节点(本原问题)。
- 2) 它是一个“或”节点，且其子节点至少有一个是可解节点。
- 3) 它是一个“与”节点，且其子节点全部是可解节点。

6. 不可解节点

关于可解节点的两个条件全部不满足的节点称为不可解节点。



7. 解树

- 解树概念:

可解节点所构成的, 并且由这些可解节点可推出初始节点 (它对应于原始问题) 为可解节点的子树称为解树。

- 在解树中一定包含初始节点。



用与/或树表示问题的步骤

- (1) 对所要求解的问题进行分解或等价变换。
- (2) 若所得的子问题不是本原问题，则继续分解或变换，直到分解或变换为本原问题。
- (3) 在分解或变换中，若是不等价的分解，则用“与树”表示，若是等价变换，则用“或树”表示。



与/或树搜索



与/或树的搜索策略

§1 与/或树的一般搜索过程

- 1) 把原始问题作为初始节点 S_0 ，并把它作为当前节点。
- 2) 应用分解或等价变换算符对当前节点进行扩展。



- 3) 为每个子节点设置指向父节点的指针。
- 4) 选择合适的子节点作为当前节点，反复执行第2)步和第3)步，在此期间要多次调用**可解标示过程**和**不可解标示过程**，直到初始节点被标示为可解节点或不可解节点为止。



§ 2 与/或树的宽度优先搜索

- 1) 把初始节点 S_0 放入OPEN表。
- 2) 把OPEN表中的第一个节点（记为节点n）取出放入CLOSED表。
- 3) 如果节点n可扩展，则：
 - ①扩展节点n，将其子节点放入OPEN表的尾部，并为每个节点配置指向父节点的指针。



②考察这些节点中是否有终止节点。若有，则标示这些终止节点为可解节点，并应用可解标示过程对其父节点、祖父节点等先辈节点中的可解节点进行标示。若初始节点 S_0 也被标示为可解节点，就得到了解树，搜索成功，退出搜索过程；若不能确定 S_0 为可解节点，则从OPEN表中删去具有可解先辈的节点。

③转第2)步。



4) 如果节点 n 不可扩展, 则:

① 标示节点 n 为不可解节点。

② 应用不可解标示过程对节点 n 的先辈节点中不可解的节点进行标示。若初始节点 S_0 也被标示为不可解节点, 则搜索失败, 表明原始问题无解, 退出搜索过程; 若不能确定 S_0 为不可解节点, 则从OPEN表中删去具有不可解先辈的节点。

③ 转第2) 步。



例：设有如图4.19所示的与/或树，节点按图中所标注的顺序号进行扩展。其中标有 t_1 、 t_2 、 t_3 、 t_4 的节点均为终止节点，A和B为不可解的端节点。

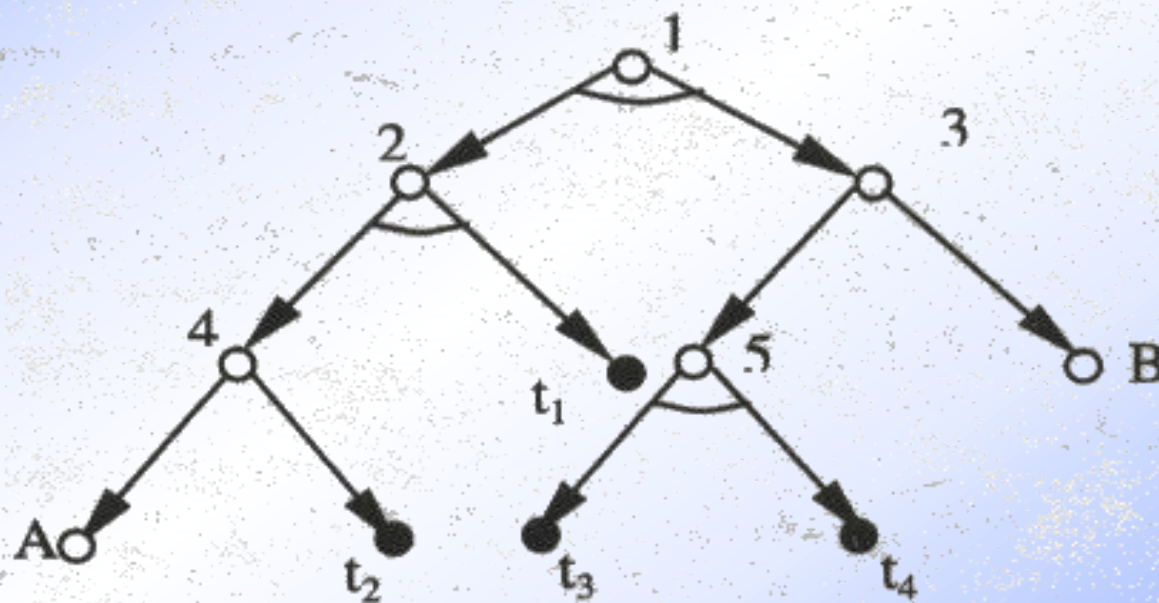


图 4.19 与/或树的宽度优先搜索



搜索过程

1) 首先扩展节点1, 得到节点2和节点3, 由于这两个子节点均不是终止节点, 所以接着扩展节点2, 此时OPEN表中只剩下节点3。

2) 扩展节点2后, 得到节点4和节点 t_1 。此时OPEN表中的节点有: 节点3、节点4及节点 t_1 。由于 t_1 是终止节点, 则标示它为可解节点, 并应用可解标示过程, 对其先辈节点中的可解节点进行标示。在此例中, t_1 的父节点2是一个“与”节点, 因此仅由 t_1 可解尚不能确定节点2是否为可解节点。所以继续搜索, 下一步扩展节点3。



3) 扩展节点3得到节点5与节点B, 两者均不是终止节点, 所以接着扩展节点4。

4) 扩展节点4后得到节点A和 t_2 。由于 t_2 是终止节点, 所以标示它为可解节点, 并应用可解标示过程标示出节点4、节点2均为可解节点, 但节点1目前还不能确定是否为可解节点。此时节点5是OPEN表中的第一个待考察的节点, 所以下一步扩展节点5。



5) 扩展节点5, 得到节点 t_3 和 t_4 。由于 t_3 和 t_4 均为终止节点, 所以被标示为可解节点, 通过应用可解标示过程可得到节点5、节点3及节点1均为可解节点。

6) 搜索成功, 得到由节点1、2、3、4、5及节点 t_1 、 t_2 、 t_3 、 t_4 构成的解树。



§3 与/或树的深度优先（有界）搜索

- 1) 把初始节点 S_0 放入OPEN表。
- 2) 把OPEN表中的第一个节点（记为节点n）取出放入CLOSED表。
- 3) 如果节点n的深度大于等于深度界限，则转第5)步的第①点。
- 4) 如果节点n可扩展，则：
 - ①扩展节点n，将其子节点放入OPEN表的首部，为每个节点配置指向父节点的指针，以备标示过程使用。



②考察这些节点中是否有终止节点。若有，则标示这些终止节点为可解节点，并应用可解标示过程对其先辈节点中的可解节点进行标示。若初始节点 S_0 也被标示为可解节点，则搜索成功，退出搜索过程；若不能确定 S_0 为可解节点，则从OPEN表中删去具有可解先辈的节点。

③转第2)步。



5) 如果节点 n 不可扩展, 则:

① 标示节点 n 为不可解节点。

② 应用不可解标示过程对节点 n 的先辈节点中不可解的节点进行标示。若初始节点 S_0 也被标示为不可解节点, 则搜索失败, 表明原始问题无解, 退出搜索过程; 若不能确定 S_0 为不可解节点, 则从OPEN表中删去具有不可解先辈的节点。

③ 转第2) 步。



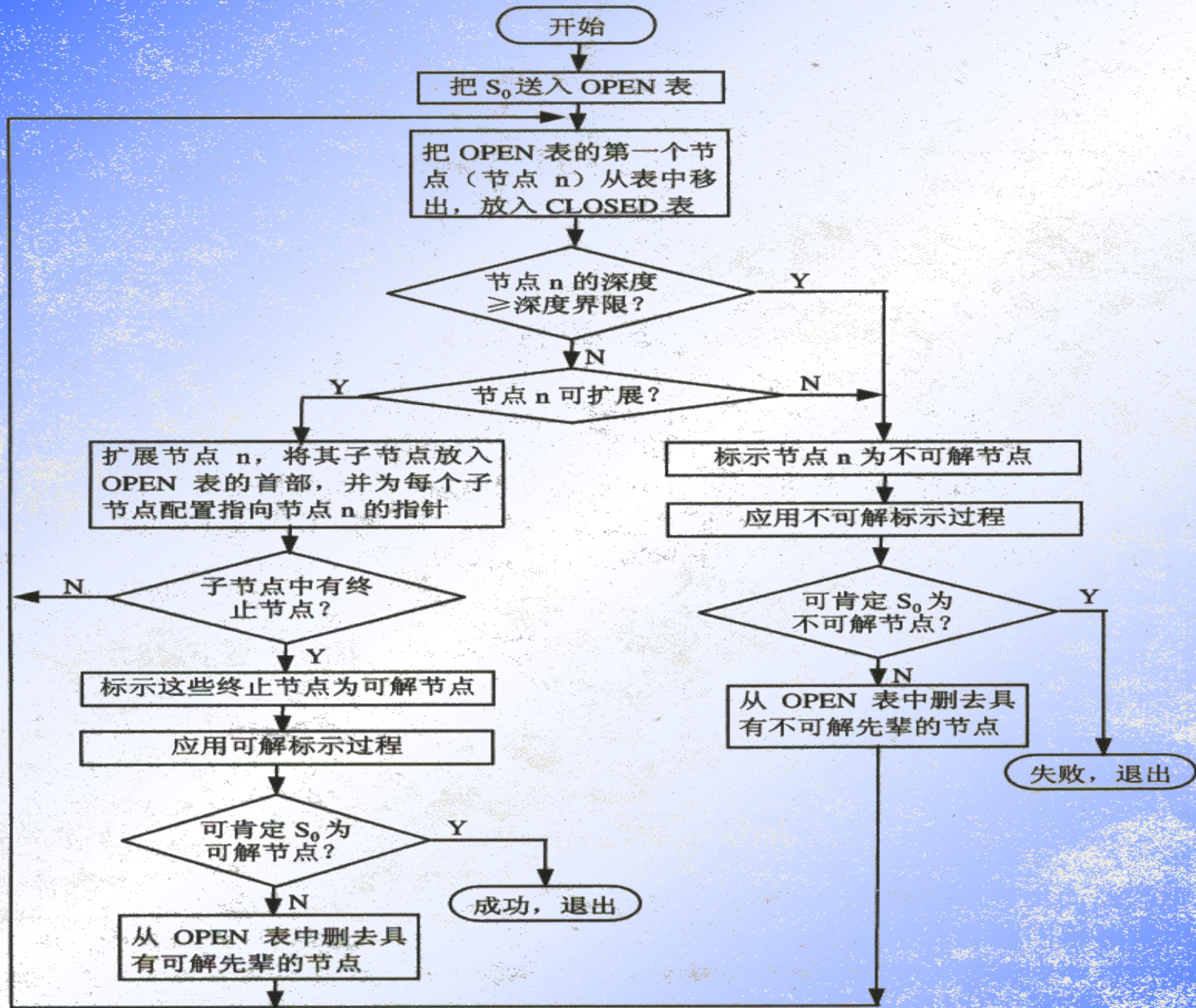


图 4.20 与/或树的有界深度优先搜索流程示意图



3.3.3 Further Examples and Applications

Example 3.3.3 MACSYMA

For **symbolically integrating** mathematical functions



Goal

$$\int \frac{x^4}{(1-x^2)^2} dx$$

$$x = \sin y$$

$$\int \frac{\sin^4 y}{\cos^4 y} dy$$

Trigonometric identity

Trigonometric identity

$$z = \tan \frac{y}{2}$$

$$\int \cot^4 y dy$$

$$\int \tan^4 y dy$$

$$\int 32 \frac{z^4}{(1+z^2)(1-z^2)^4} dz$$

$$z = \cot y$$

$$z = \tan y$$

$$\int -\frac{dz}{z^4(1+z^2)}$$

$$\int \frac{z^4}{1+z^4} dz$$

Divide numerator by denominator

$$\int -dz$$

$$\int \left(-1 + z^2 + \frac{1}{1+z^2}\right) dz$$

$$\int \frac{dz}{1+z^2}$$

$$z = \tan w$$

$$\int dz$$

$$\int z^2 dz$$

$$\int dw$$

Example 3.3.4

Goal-driven **and / or** graph search



1. Fred is a collie.
collie(fred).

// **collie**: 牧羊犬

2. Sam is Fred's master.
master(fred,sam).

3. The day is Saturday.
day(saturday).

4. It is cold on Saturday.
 \neg (**warm(saturday)**).

5. Fred is trained.
trained(fred).

//**spaniel**: 西班牙猎犬

6. Spaniels are good dogs and so are trained collies.
 $\forall X[\text{spaniel}(X) \vee (\text{collie}(X) \wedge \text{trained}(X)) \rightarrow \text{gooddog}(X)]$

7. If a dog is a good dog and has a master then he will be with his master.
 $\forall (X,Y,Z) [\text{gooddog}(X) \wedge \text{master}(X,Y) \wedge \text{location}(Y,Z) \rightarrow \text{location}(X,Z)]$

8. If it is Saturday and warm, then Sam is at the park.
 $(\text{day(saturday)} \wedge \text{warm(saturday)}) \rightarrow \text{location(sam,park)}.$

9. If it is Saturday and not warm, then Sam is at the museum.
 $(\text{day(saturday)} \wedge \neg (\text{warm(saturday)})) \rightarrow \text{location(sam,museum)}.$



● Facts

1. collie(fred)
2. master(fred, sam)
3. day(saturday)
4. \neg (warm(saturday))
5. trained(fred)

// collie: 牧羊犬

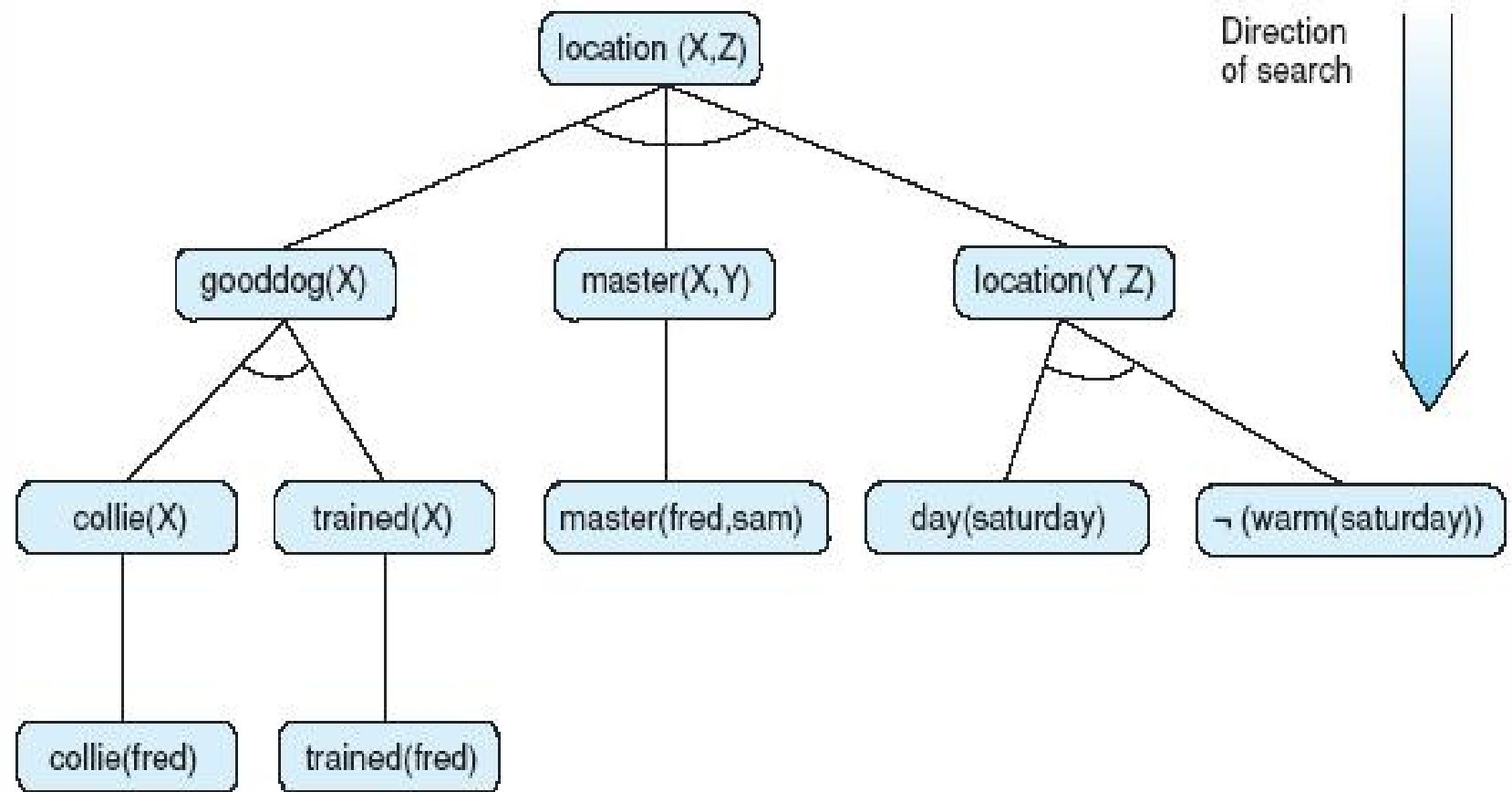
● rules

//spaniel: 西班牙猎犬

6. $\forall X[\text{spaniel}(X) \vee (\text{collie}(X) \wedge \text{trained}(X)) \rightarrow \text{gooddog}(X)]$
7. $\forall (X,Y,Z)[\text{gooddog}(X) \wedge \text{master}(X,Y) \wedge \text{location}(Y,Z) \rightarrow$
 $\text{location}(X,Z)]$
8. $(\text{day}(\text{saturday}) \wedge \text{warm}(\text{saturday})) \rightarrow \text{location}(\text{sam}, \text{park})$
9. $(\text{day}(\text{saturday})) \wedge \neg(\text{warm}(\text{saturday})) \rightarrow$
 $\text{location}(\text{sam}, \text{museum})$

- Goal : $\exists X \text{ location}(\text{fred}, X)$





Substitutions = {fred/X, sam/Y, museum/Z}



Example 3.3.5

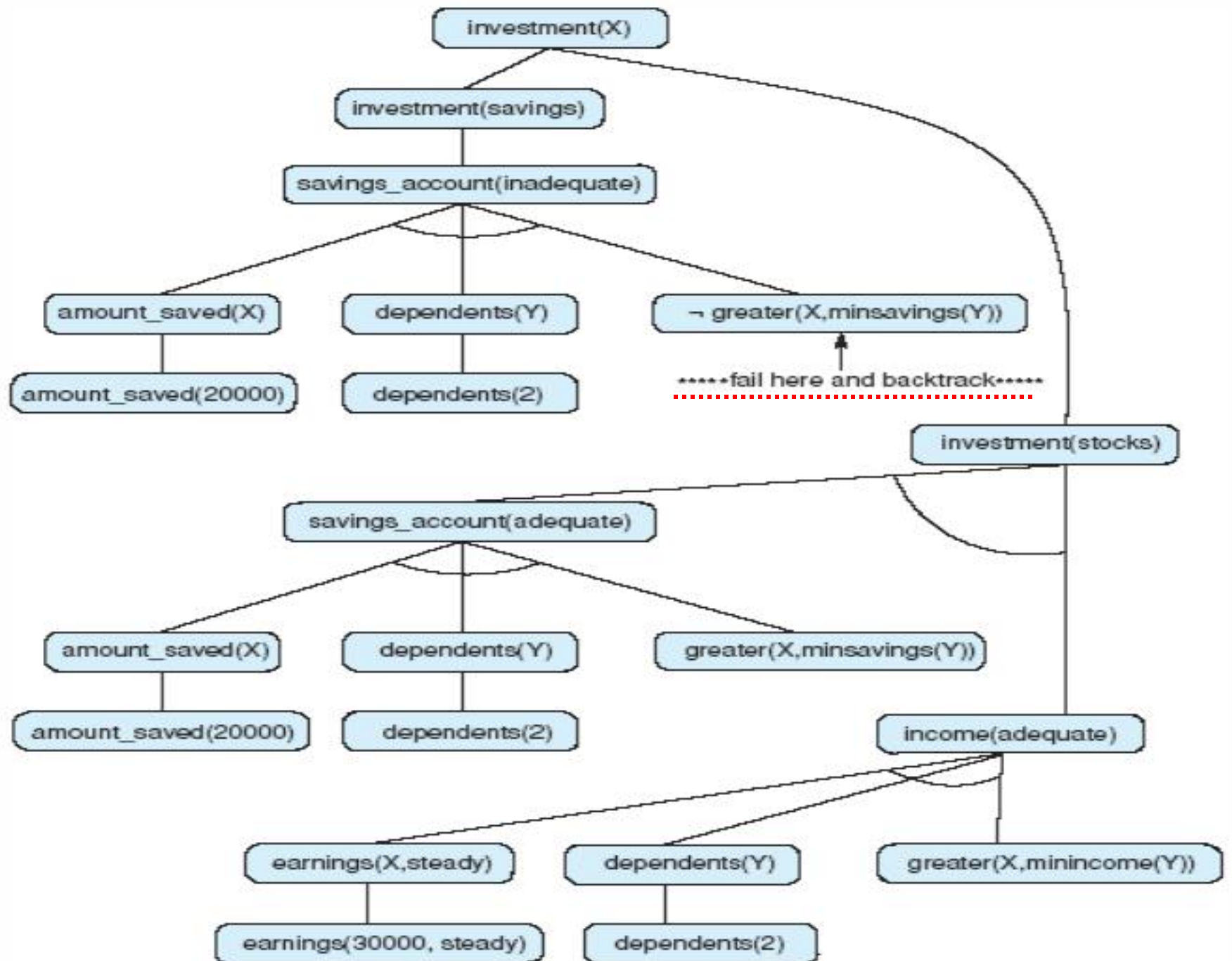
The Financial Advisor Revisited

1. $\text{saving_account}(\text{inadequate}) \rightarrow \text{investment}(\text{savings})$
2. $\text{saving_account}(\text{adequate}) \wedge \text{income}(\text{adequate})$
 $\rightarrow \text{investment}(\text{stocks})$
3. $\text{saving_account}(\text{adequate}) \wedge \text{income}(\text{inadequate})$
 $\rightarrow \text{investment}(\text{combination})$
4. $\forall X \text{ amount_saved}(X) \wedge \exists Y(\text{dependents}(Y)$
 $\wedge \text{greater}(X, \text{minsavings}(Y)))$
 $\rightarrow \text{saving_account}(\text{adequate})$
5. $\forall X \text{ amount_saved}(X) \wedge \exists Y(\text{dependents}(Y)$
 $\wedge \neg \text{greater}(X, \text{minsavings}(Y)))$
 $\rightarrow \text{saving_account}(\text{inadequate})$



6. $\forall X \text{ earnings}(X, \text{steady}) \wedge \exists Y(\text{dependents}(Y) \wedge \text{greater}(X, \text{minincome}(Y))) \rightarrow \text{income}(\text{adequate})$
7. $\forall X \text{ earnings}(X, \text{steady}) \wedge \exists Y(\text{dependents}(Y) \wedge \neg \text{greater}(X, \text{minincome}(Y))) \rightarrow \text{income}(\text{inadequate})$
8. $\forall X \text{ earnings}(X, \text{unsteady}) \rightarrow \text{income}(\text{inadequate})$
9. $\text{amount_saved}(20000)$
10. $\text{earnings}(30000, \text{steady})$
11. $\text{dependents}(2)$
- **Goal** : $\exists X \text{ investment}(X)$





Example 3.3.6

An English Language **Parser and
Sentence **Generator****



A simple Subset of **English Grammar**

1. **A sentence is a noun phrase followed by a verb phrase.**
sentence \leftrightarrow np vp
2. **A noun phrase is a noun.**
np \leftrightarrow n
3. **A noun phrase is an article followed by a noun.**
np \leftrightarrow art n
4. **A verb phrase is a verb.**
vp \leftrightarrow v
5. **A verb phrase is a verb followed by a noun phrase.**
vp \leftrightarrow v np
6. **art \leftrightarrow a**
7. **art \leftrightarrow the**
(“a” and “the” are articles)
8. **n \leftrightarrow man**
9. **n \leftrightarrow dog**
(“man” and “dog” are nouns)
10. **v \leftrightarrow likes**
11. **v \leftrightarrow bites**
(“likes” and “bites” are verbs)

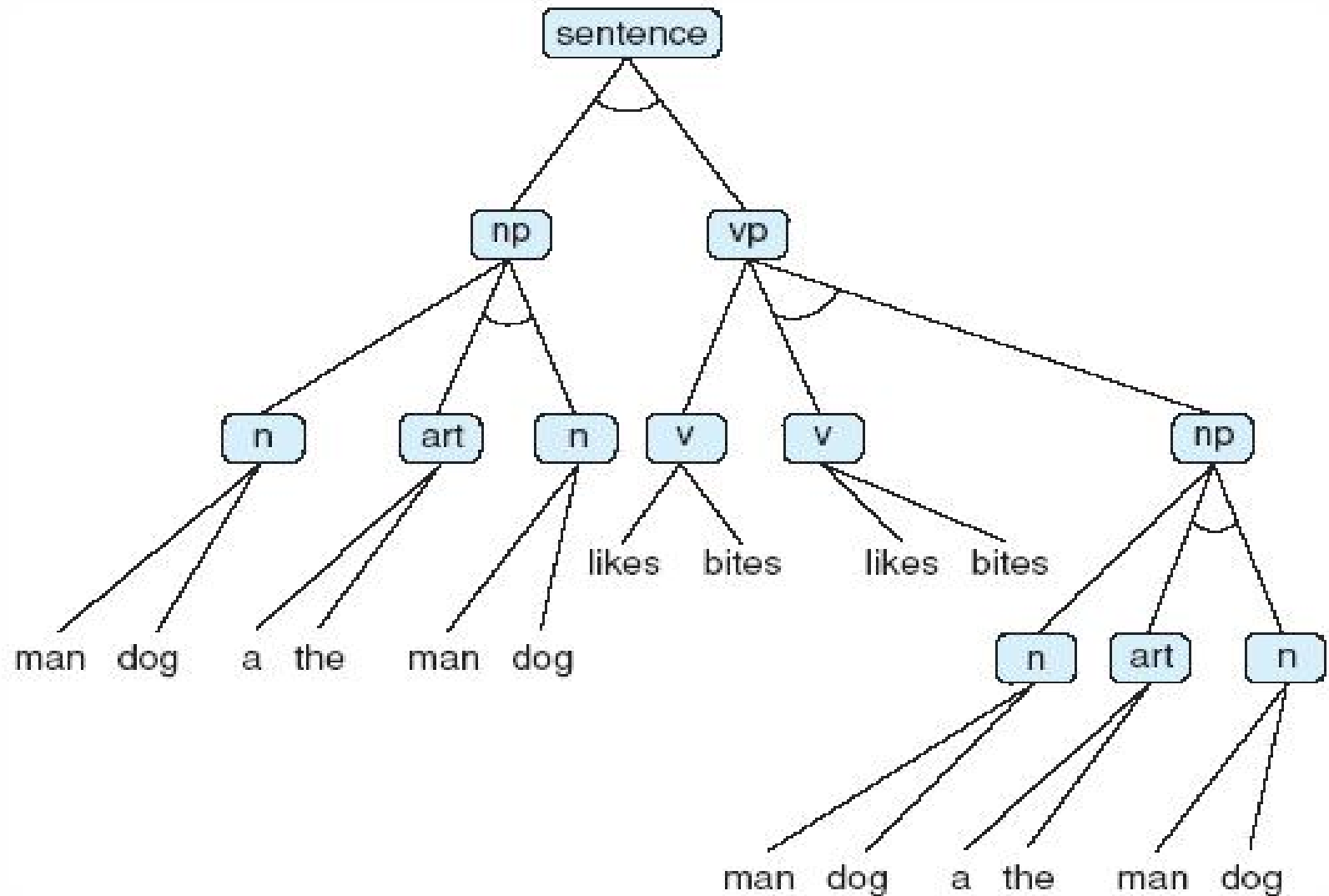


A simple Subset of **English Grammar**

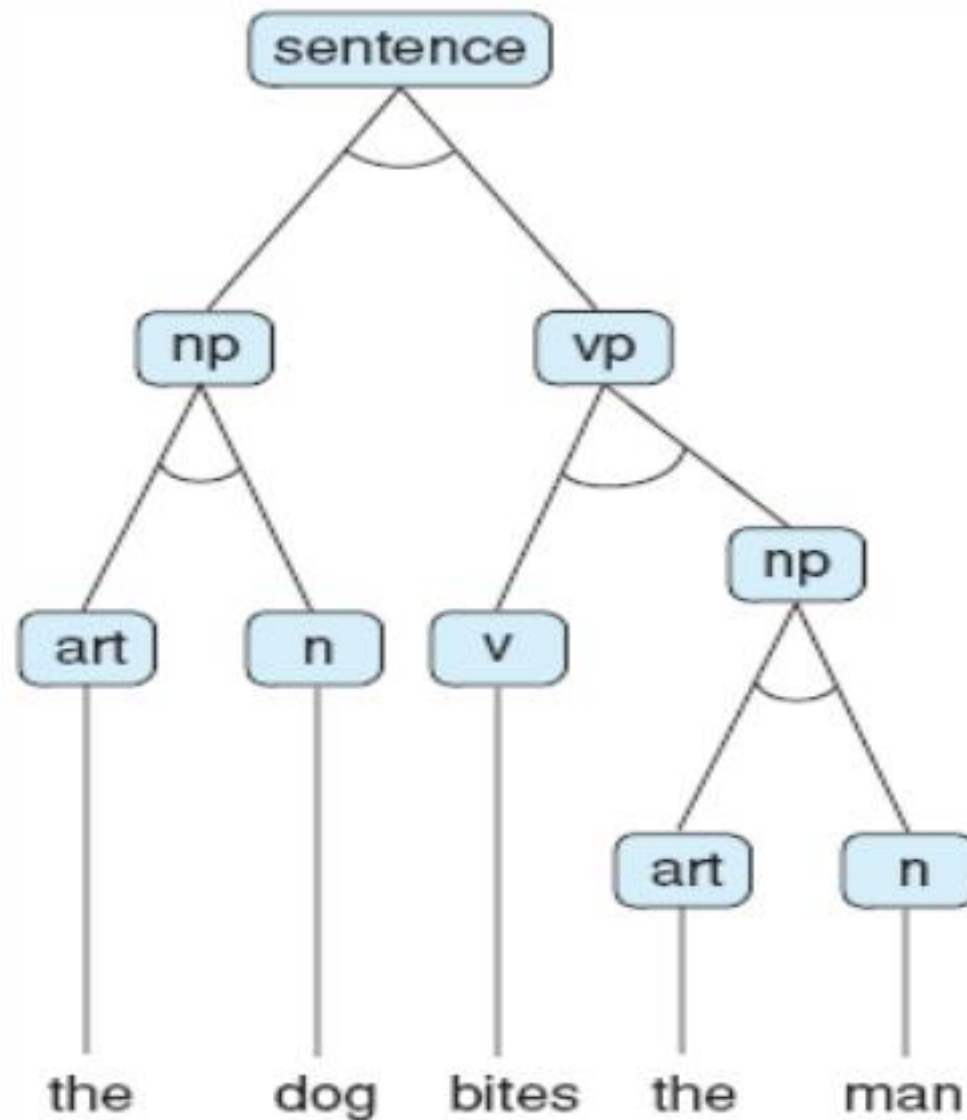
1. **sentence** \longleftrightarrow **np vp**
2. **np** \longleftrightarrow **n**
3. **np** \longleftrightarrow **art n**
4. **vp** \longleftrightarrow **v**
5. **vp** \longleftrightarrow **v np**
6. **art** \longleftrightarrow **a**
7. **art** \longleftrightarrow **the**
8. **n** \longleftrightarrow **man**
9. **n** \longleftrightarrow **dog**
10. **v** \longleftrightarrow **likes**
11. **v** \longleftrightarrow **bites**



And / or graph for the grammar



Parse tree for the sentence “The dog bites the man”



Exercises P. 121

2.

4.

5.

7.

10.

12.



Hint for Exercises P. 121

2.

Hint:

(1) State description: (F, W, G, C)

(2) Initial state: (1, 1, 1, 1)

(3) legal moves (operators, rules) :

FF+ , FW- ,

(4) Breadth-First-Search required

5.

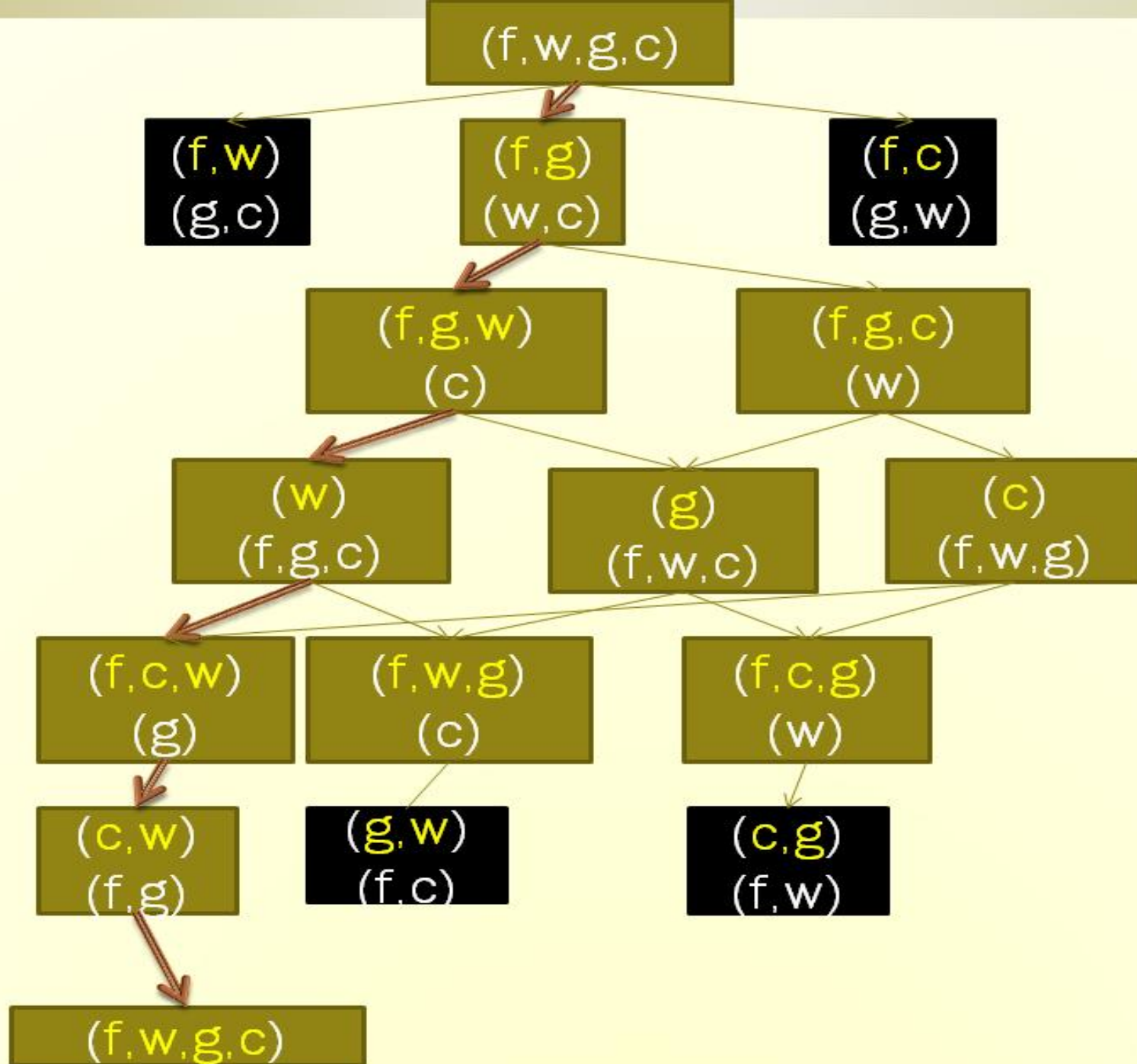
Hint: We assume the goal state is G

10.

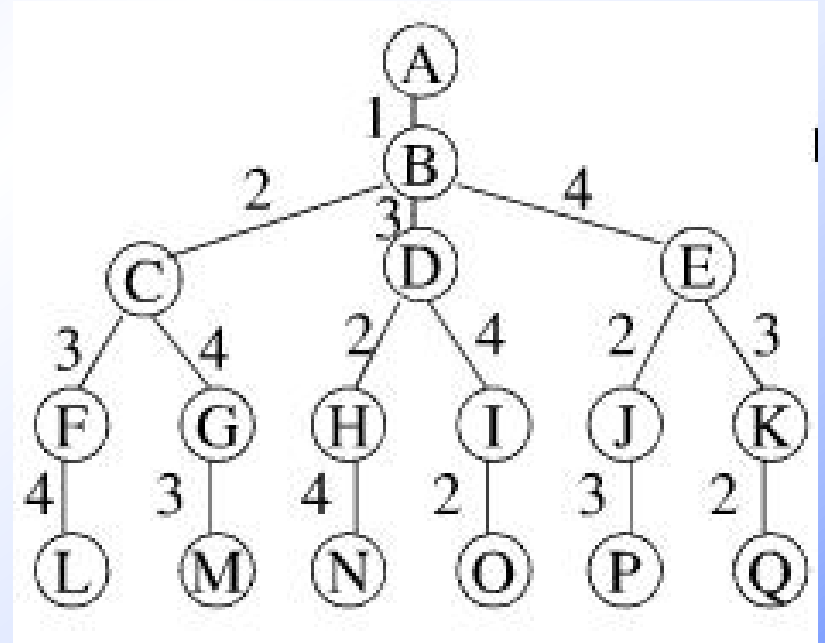
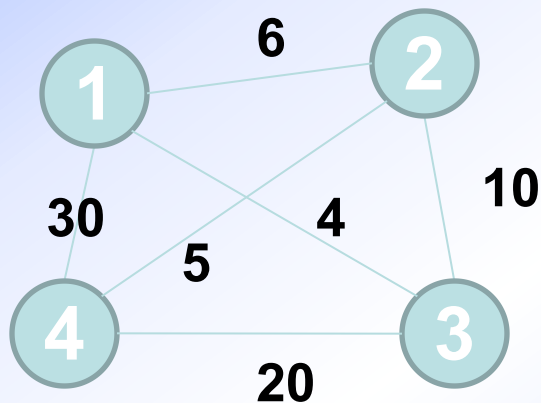


2. Give the graph representation for the farmer, wolf, goat, and cabbage problem of figures in the Prolog supplementary programming material. Let the nodes represent states of the world; e.g., the farmer and the goat are on the west bank and the wolf and cabbage on the east. Discuss the advantages of each alternative representation at the beginning of Chapter 3 for searching this space.





4. Give an instance of the traveling salesperson problem for which the nearest-neighbor strategy fails to find an optimal path. Suggest another heuristic for this problem.



5. “Hand run” the backtrack algorithm on the graph in Figure 3.29. Begin from state A. Keep track of the successive values of NSL, SL, CS, etc.

- **SL**: 状态列表，列出当前正在实验路径的状态。如果发现了目标，那么**SL**变包含了解路径上状态的有序列表；
- **NSL**: 新状态列表，含有等待评估的结点，也就是其后代还没有被产生和搜索的结点。
- **DE**: 记录死端，列出已经发现其后代不包含目标的状态。
- **CS**: 当前状态；



- We assume there is no goal state in the search space.
Initialize:
- $SL = [A]$; $NSL = [A]$; $DE = []$; $CS = A$;

LOOP	CS	SL	NSL	DE
0	A	[A]	[A]	[]
1	B	[B A]	[B C D A]	[]
2	E	[E B A]	[E F G B C D A]	[]
3	J	[J E B A]	[J K L E F G B C D A]	[]
4	K	[K E B A]	[K L E F G B C D A]	[J]
5	L	[L E B A]	[L E F G B C D A]	[K J]
6	F	[F B A]	[F G B C D A]	[E L K J]
7	G	[G B A]	[G B C D A]	[F E L K J]
8	M	[M G B A]	[M N H G B C D A]	[F E L K J]
9	N	[N G B A]	[N H G B C D A]	[M F E L K J]
10	H	[H G B A]	[H G B C D A]	[N M F E L K J]
11	O	[O H G B A]	[O P H G B C D A]	[N M F E L K J]
12	P	[P H G B A]	[M N H G B C D A]	[O N M F E L K J]
13	C	[C A]	[C D A]	[G H P O N M F E L K J]
14	D	[D A]	[D A]	[C G H P O N M F E L K J]
15	I	[I D A]	[I D A]	[C G H P O N M F E L K J]
16	R	[R I D A]	[R I D A]	[C G H P O N M F E L K J]
17	-	[]	[]	[A D I R C G H P O N M F E L K J]



7. Determine whether goal- or data-driven search would be preferable for solving each of the following problems. Justify your answer.

- **数据驱动**是从问题的事实入手，然后应用规则或合法移动来产生通往目标的新事实；
- **目标驱动**推理是把焦点集中在目标上，寻找可以产生这个目标的规则，并环环相扣地反向追索相继的规则和子目标直到到达问题中给定的事实。



▶ 适用于目标驱动搜索的情况：

1. **目标或假设是在问题陈述中给出的，或者他们很容易被形式化。**例如，在数学定理证明程序中，目标是要证明的定理，很多诊断系统以一种系统的方式考虑可能的诊断，并利用目标驱动推理或者排除这些可能的诊断。
 2. **与问题事实匹配的规则数量非常多，因而产生的结论或目标也越来越多。**因为较早的选择目标可以有效排除大量分支，所以目标驱动搜索可以更高效的检出这些空间。例如，在数学定理证明程序中，用来产生给定定理的规则数远远小于可以应用到整个公理集的规则数。
 3. **问题数据不是给定的，而是要问题求解器获取的。**在这种情况下，目标驱动搜索可以引导如何获取数据。例如在医疗诊断程序中，现有的诊断化验种类非常多，医生建议那些对确认或者排除某个假设有必要的化验。
- ▶ 目标驱动是使用关于预期目标的知识来引导搜索，通过相关的规则来排除空间分支。



- 适用于数据驱动：

1. 问题的初始陈述给出了所有或大部分数据。解释问题经常符合这种情况，给出一系列数据并要求系统提供一个高层的解释。分析特定数据的系统适合采用数据驱动方法，比如勘探程序，他们对地质数据进行解释，或者分析某个地方可能蕴藏什么矿。
 2. 潜在目标的数量非常庞大，但是使用特定问题实例的给定信息和事实的方式很有限。例如dendral程序，它是一个用来分析有机化合物分子结构的专家系统，这个系统的分析依据是化合物的分子式、大规模的光谱数据以及其他化学知识。对任何一种有机化合物来说，都存在数量及其客观的可能结构。然而，大规模的化合物光谱数据使dendgal可以排除这些可能结构中的大多数，仅留下其中的很少几个结构。
 3. 难以组成目标或假设。例如上边的例子，初始时可能对化合物的可能结构知之甚少。
- 数据驱动使用了从一个问题的已知数据中提出的知识和约束，引导搜索沿着已知的路线到正解。



1. 诊断汽车的机械故障：**目标驱动**。根据汽车表现出的状态（目标）排除或者确认可能的故障。
2. **双向搜索**：目标驱动：打听下john的后代；数据驱动：找一找你和这个所谓远方表妹的祖辈；看看有多少个共同的人。
3. **数据驱动**：根据你和这位远方堂弟的现有的亲属关系，分别推测你们的祖辈，看是否有相同的祖辈。
4. 平面几何定理证明：**目标驱动**。
5. **数据驱动**，程序输入大量数据,规则少。
6. **数据驱动**，目标太多。



10. Trace the good-dog problem of Example 3.3.4 in a data-driven fashion.

事实1 fred/X in 规则6;

事实2 fred/x, sam/y, in规则7;

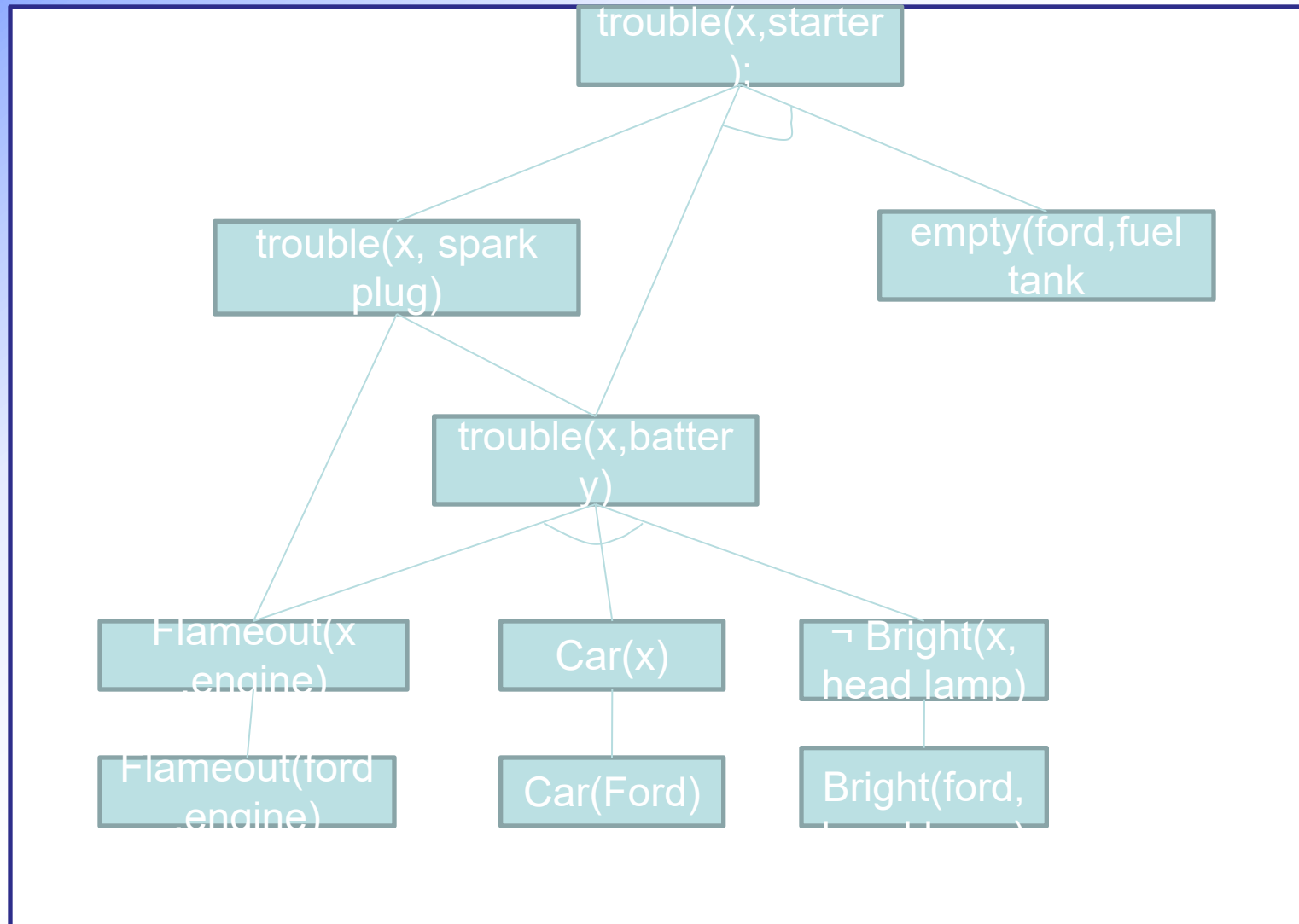
事实3和4 满足规则9的前提, 所以加入新的知识,
location (sam, museum) ;

事实5和1满足规则6前提, 所以加入新知识, gooddog
(fred) ;

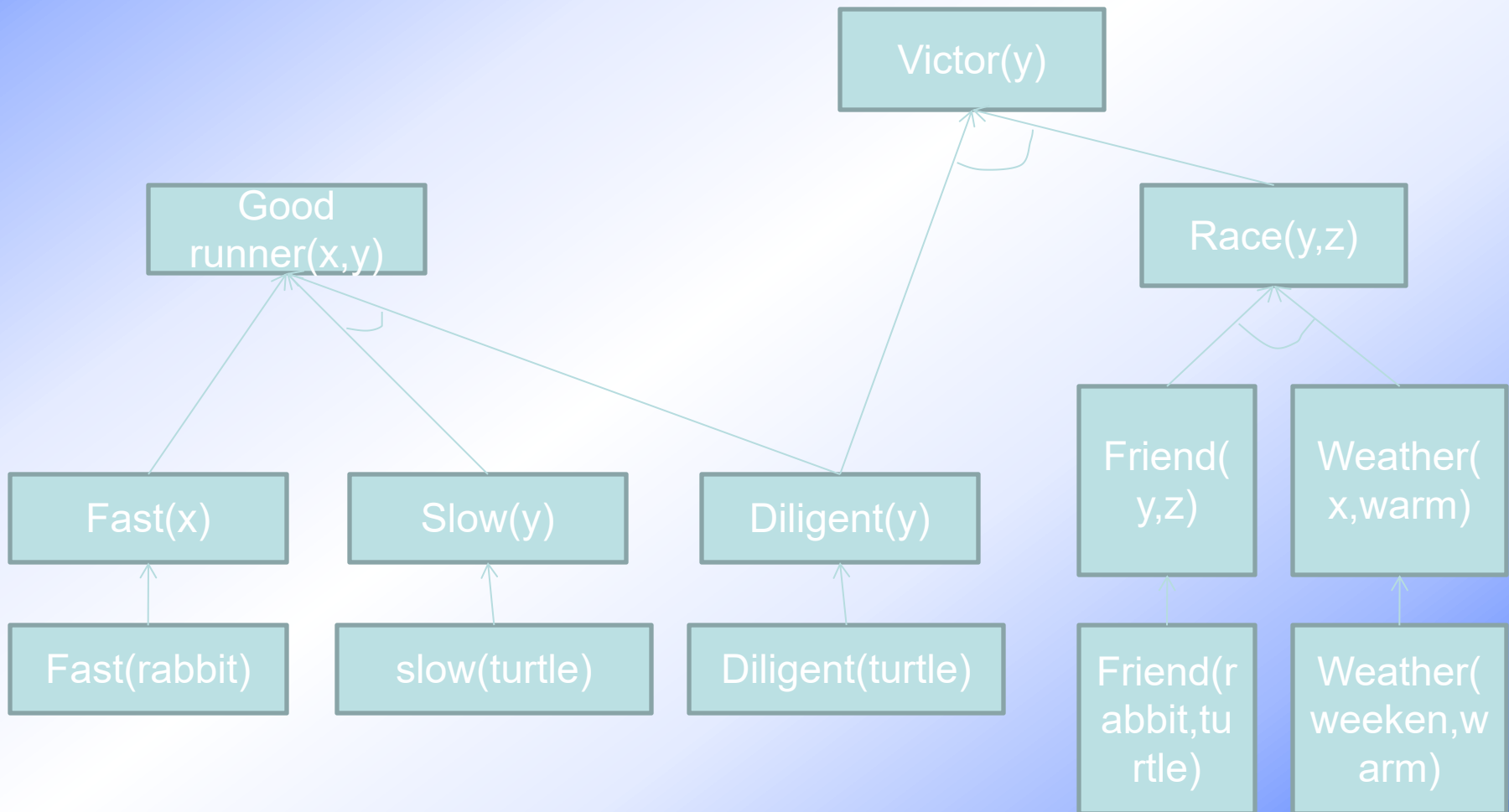
规则7条件激活, 所以得到答案。在博物馆。



11. Give another example of an and/or graph search problem. Develop part of the search space.



11. Give another example of an and/or graph search problem. Develop part of the search space



13. Add rules defining adjectives and adverbs to the grammar of Example 3.3.6.

Np \leftrightarrow adj n

np \leftrightarrow art adj n

np \leftrightarrow adj adj n

np \leftrightarrow art adj adj n

vp \leftrightarrow v adv

vp \leftrightarrow v adv np etc.

adj \leftrightarrow big

adj \leftrightarrow ugly

adj \leftrightarrow mean

adv \leftrightarrow easily

adv \leftrightarrow fiercely etc.



Chapter Three Exercises

1.

John is going on a trip with an extra suitcase that he plans to fill with souvenirs. Each souvenir has a weight and a value. The weight limit on checked baggage is 50 pounds. Describe the state space representation of the problem that maximizes the value of the souvenirs John brings home, while minimizing their total weight so as not to exceed the weight limit.



2.

Write the propositional calculus rules for the and/or graph given in Figure 3.24.



Goal

$$\int \frac{x^4}{(1-x^2)^2} dx$$

$$x = \sin y$$

$$\int \frac{\sin^4 y}{\cos^4 y} dy$$

Trigonometric identity

Trigonometric identity

$$z = \tan \frac{y}{2}$$

$$\int \cot^4 y dy$$

$$\int \tan^4 y dy$$

$$\int 32 \frac{z^4}{(1+z^2)(1-z^2)^4} dz$$

$$z = \cot y$$

$$z = \tan y$$

$$\int -\frac{dz}{z^4(1+z^2)}$$

$$\int \frac{z^4}{1+z^4} dz$$

Divide numerator by denominator

$$\int -dz$$

$$\int \left(-1 + z^2 + \frac{1}{1+z^2}\right) dz$$

$$\int \frac{dz}{1+z^2}$$

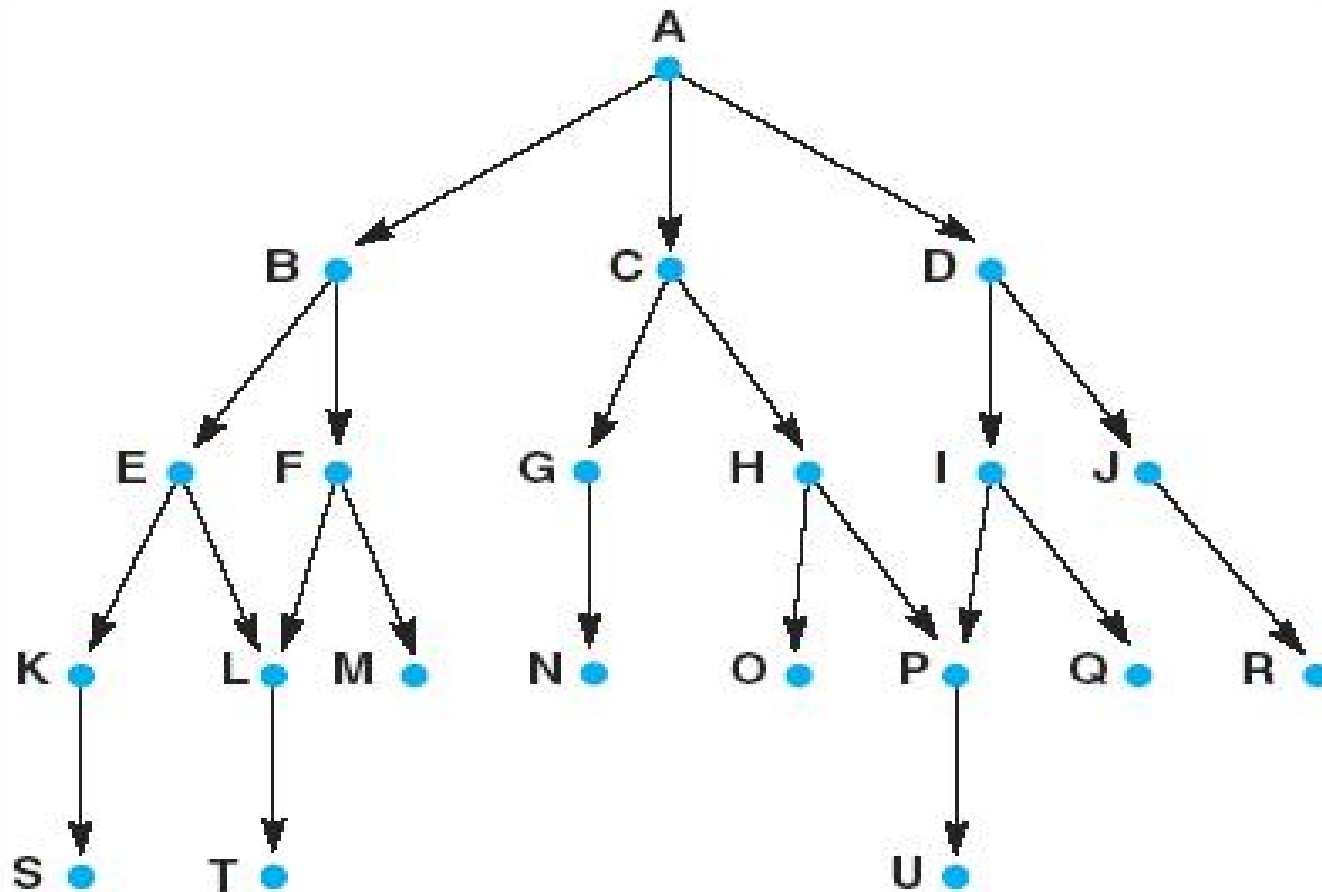
$$z = \tan w$$

$$\int dz$$

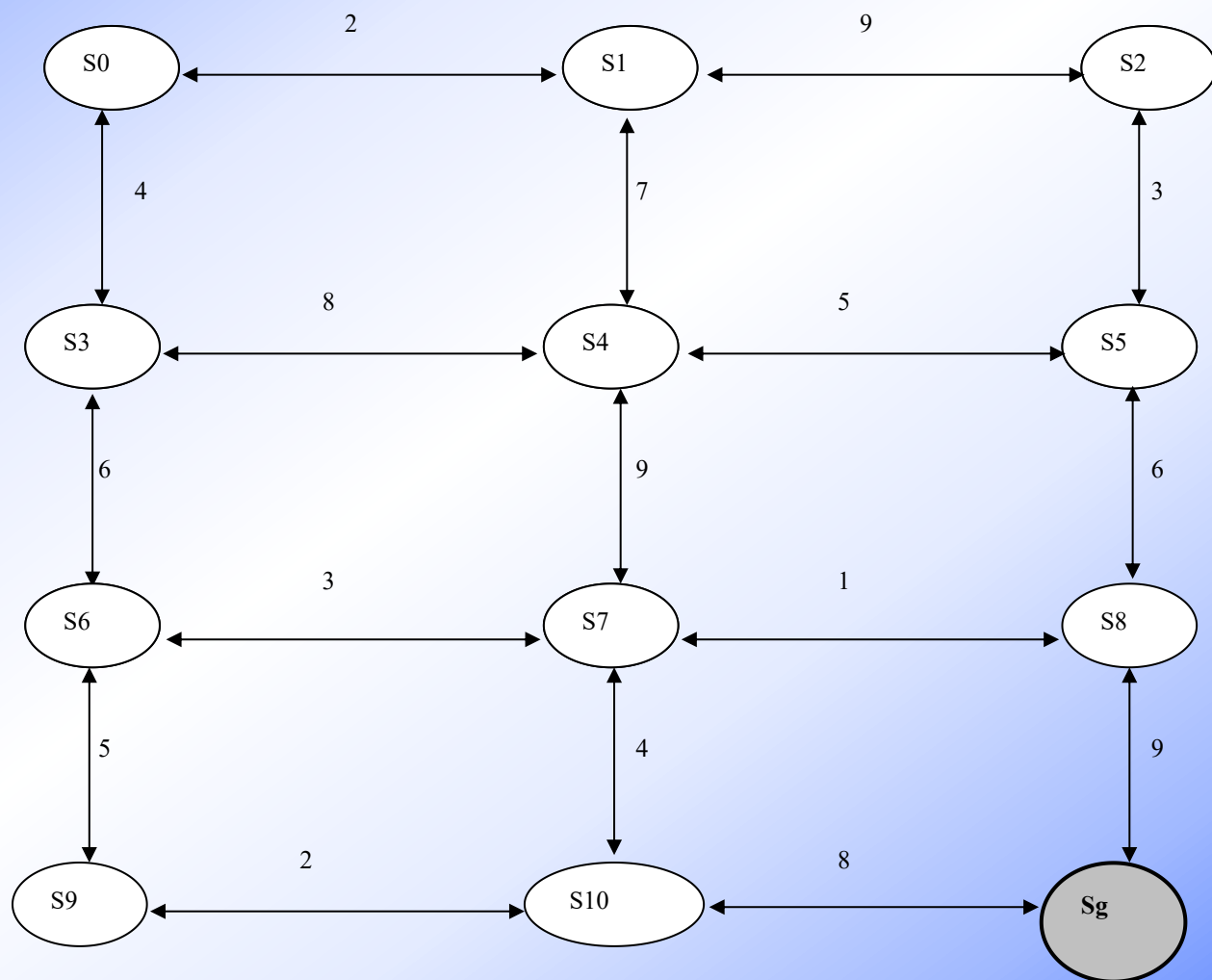
$$\int z^2 dz$$

$$\int dw$$

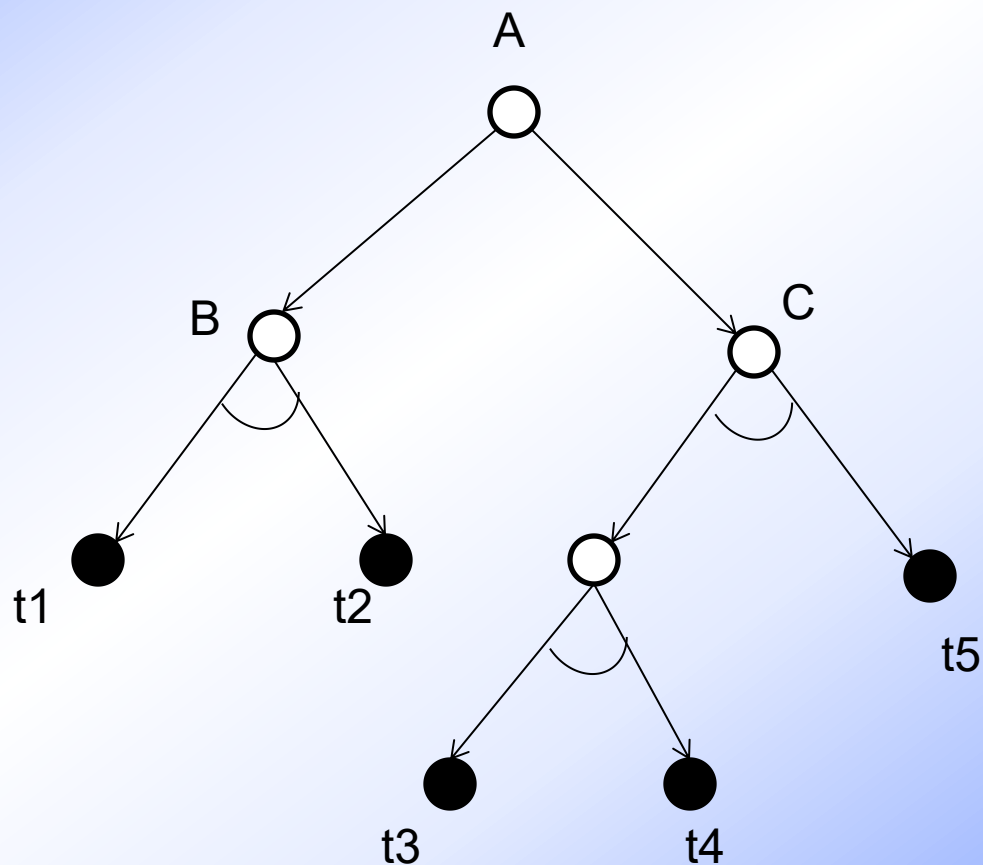
3. “Hand run” the backtrack algorithm on the graph in Figure 3.15. Begin from state A. Keep track of the successive values of NSL, SL, CS, etc.



4. 已知状态图如下，其中 S_0 为初始状态， S_g 为目标状态，每条弧旁边的数字是所对应操作的代价。请求最佳解，并给出搜索过程图示。



5.对下图分别应用与/或树的宽度优先搜索和与/或树的有界深度度优先搜索求出解树。



1.Solution:

N: Each node in the graph contains the current value of souvenirs in the suitcase, the total current weight of the souvenirs, the maximum value the suitcase can hold, and the maximum value of souvenirs in the suitcase found so far.

A: Each arc represents the value of adding a given souvenir, and the weight of a given souvenir to be subtracted from the available weight.

S: The start state is an empty suitcase.

GD: The goal description is a packed suitcase in which the souvenirs have the maximum value, with the total weight not to exceed the weight limit.



2.Solution:

$$\int dw \leftrightarrow \int \frac{dz}{1+z^2}$$

$$\int dz \leftrightarrow \int -dz$$

$$\int -dz \wedge \int z^2 dz \wedge \int \frac{dz}{1+z^2} \rightarrow \int \left(-1 + z^2 + \frac{1}{1+z^2} \right) dz$$

$$\int \left(-1 + z^2 + \frac{1}{1+z^2} \right) dz \leftrightarrow \int \frac{z^4}{1+z^4} dz$$

$$\int \frac{z^4}{1+z^4} dz \leftrightarrow \int \tan^4 y dy$$

$$\int -\frac{dz}{z^4(1+z^2)} \leftrightarrow \int \cot^{-4} y dy$$

$$\int \cot^{-4} y dy \vee \int \tan^4 y dy \vee \int 32 \frac{z^4}{(1+z^2)(1-z^2)^4} dz \rightarrow \int \frac{\sin^4 y}{\cos^4 y} dy$$

$$\int \frac{\sin^4 y}{\cos^4 y} dy \leftrightarrow \int \frac{x^4}{(1-x^2)^2} dx$$




Iteration	CS	SL	NSL	DE
0	A	[A]	[A]	[]
1	B	[BA]	[BCDA]	[]
2	E	[EBA]	[EFBCDA]	[]
3	K	[KEBA]	[KLEFBCDA]	[]
4	S	[SKEBA]	[SKLEFBCDA]	[]
5	L	[LEBA]	[LEFBCDA]	[KS]
6	T	[TLEBA]	[TLEFBCDA]	[KS]
7	F	[FBA]	[FBCDA]	[ELTKS]
8	M	[MFBA]	[MFBCDA]	[ELTKS]
9	C	[CA]	[CDA]	[BFMELTKS]
10	G	[GCA]	[GHCDA]	[BFMELTKS]



11	N	[NGCA]	[NGHCDA]	[BFMELTKS]
12	H	[HCA]	[HCDA]	[GNBFMELTKS]
13	O	[OHCA]	[OPHCDA]	[GNBFMELTKS]
14	P	[PHCA]	[PHCDA]	[OGNBFMELTKS]
15	U	[UPHCA]	[UPHCDA]	[OGNBFMELTKS]
16	D	[DA]	[DA]	[CHPUOGNBFMELTKS]
17	I	[IDA]	[IJDA]	[CHPUOGNBFMELTKS]
18	Q	[QIDA]	[QIJDA]	[CHPUOGNBFMELTKS]
19	J	[JDA]	[JDA]	[IQCHPUOGNBFMELTKS]
20	R	[RJDA]	[RJDA]	[IQCHPUOGNBFMELTKS]
21	-	[]	[]	[ADJRIQCHPUOGNBFMELTKS]



传教士与野人问题



应用举例

- 传教士与野人问题描述：

有**3**名传教士和**3**名野人来到一条河的左岸，欲乘一条船渡河到右岸，该船的最大负载能力为**2**人，传教士与野人均可撑船。在任何时候，不论是在右岸还是左岸，如果野人人数超过传教士人数，那么，野人就会吃掉传教士。为了规划出一个渡河方案，把**6**个人都安全地渡过河去（请用产生式表示法表示求解该问题的所有规则，并给出搜索的状态空间图）。



问题解答：(1) 综合数据库

解:用三元组表示左岸的状态, 即 (M_L, C_L, B_L) , 其中

$$0 \leq M_L, C_L \leq 3, B_L \in \{0, 1\}$$

其中M、C分别代表某一岸上传教士与野人的数目, $B=1$ 表示船在这一岸, $B=0$ 则表示船不在。

此时问题描述简化为: $(3, 3, 1) \rightarrow (0, 0, 0)$ 。

对于 **$N=3$** 的 **$M-C$** 问题, 状态空间的总数为 $4 \times 4 \times 2 = 32$, 根据约束条件的要求, 可以看出只有20个合法状态。再进一步分析后, 又发现有4个合法状态实际上是不可能达到的。因此实际的问题空间仅有**16**种状态符合要求, 其它的状态不符合要求。

下表列出分析的结果:



(M_L, C_L, B_L)	(M_L, C_L, B_L)
$(0\ 0\ 1)$ 达不到	$(0\ 0\ 0)$
$(0\ 1\ 1)$	$(0\ 1\ 0)$
$(0\ 2\ 1)$	$(0\ 2\ 0)$
$(0\ 3\ 1)$	$(0\ 3\ 0)$ 达不到
$(1\ 0\ 1)$ 不合法	$(1\ 0\ 0)$ 不合法
$(1\ 1\ 1)$	$(1\ 1\ 0)$
$(1\ 2\ 1)$ 不合法	$(1\ 2\ 0)$ 不合法
$(1\ 3\ 1)$ 不合法	$(1\ 3\ 0)$ 不合法
$(2\ 0\ 1)$ 不合法	$(2\ 0\ 0)$ 不合法
$(2\ 1\ 1)$ 不合法	$(2\ 1\ 0)$ 不合法
$(2\ 2\ 1)$	$(2\ 2\ 0)$
$(2\ 3\ 1)$ 不合法	$(2\ 3\ 0)$ 不合法
$(3\ 0\ 1)$ 达不到	$(3\ 0\ 0)$
$(3\ 1\ 1)$	$(3\ 1\ 0)$
$(3\ 2\ 1)$	$(3\ 2\ 0)$
$(3\ 3\ 1)$	$(3\ 3\ 0)$ 达不到



(2) 规则集合:

由摆渡操作组成。

该问题主要有两种操作:

p_{mc} 操作(规定为从左岸划向右岸),

q_{mc} 操作(从右岸划向左岸)。

每次摆渡操作, 船上人数有五种组合, 因而组成有**10**条规则的集合。下面定义的规则前**5**条为 p_{mc} 操作(从左岸划向右岸), 后**5**条为 q_{mc} 操作(从右岸划向左岸)。



规则集合

- R1: if $(M_L, C_L, B_L=1)$ then (M_L-1, C_L, B_L-1) ; (p_{10} 操作)
- R2: if $(M_L, C_L, B_L=1)$ then (M_L, C_L-1, B_L-1) ; (p_{01} 操作)
- R3: if $(M_L, C_L, B_L=1)$ then (M_L-1, C_L-1, B_L-1) ; (p_{11} 操作)
- R4: if $(M_L, C_L, B_L=1)$ then (M_L-2, C_L, B_L-1) ; (p_{20} 操作)
- R5: if $(M_L, C_L, B_L=1)$ then (M_L, C_L-2, B_L-1) ; (p_{02} 操作)
- R6: if $(M_L, C_L, B_L=0)$ then (M_L+1, C_L, B_L+1) ; (q_{10} 操作)
- R7: if $(M_L, C_L, B_L=0)$ then (M_L, C_L+1, B_L+1) ; (q_{01} 操作)
- R8: if $(M_L, C_L, B_L=0)$ then (M_L+1, C_L+1, B_L+1) ; (q_{11} 操作)
- R9: if $(M_L, C_L, B_L=0)$ then (M_L+2, C_L, B_L+1) ; (q_{20} 操作)
- R10: if $(M_L, C_L, B_L=0)$ then (M_L, C_L+2, B_L+1) ; (q_{02} 操作)



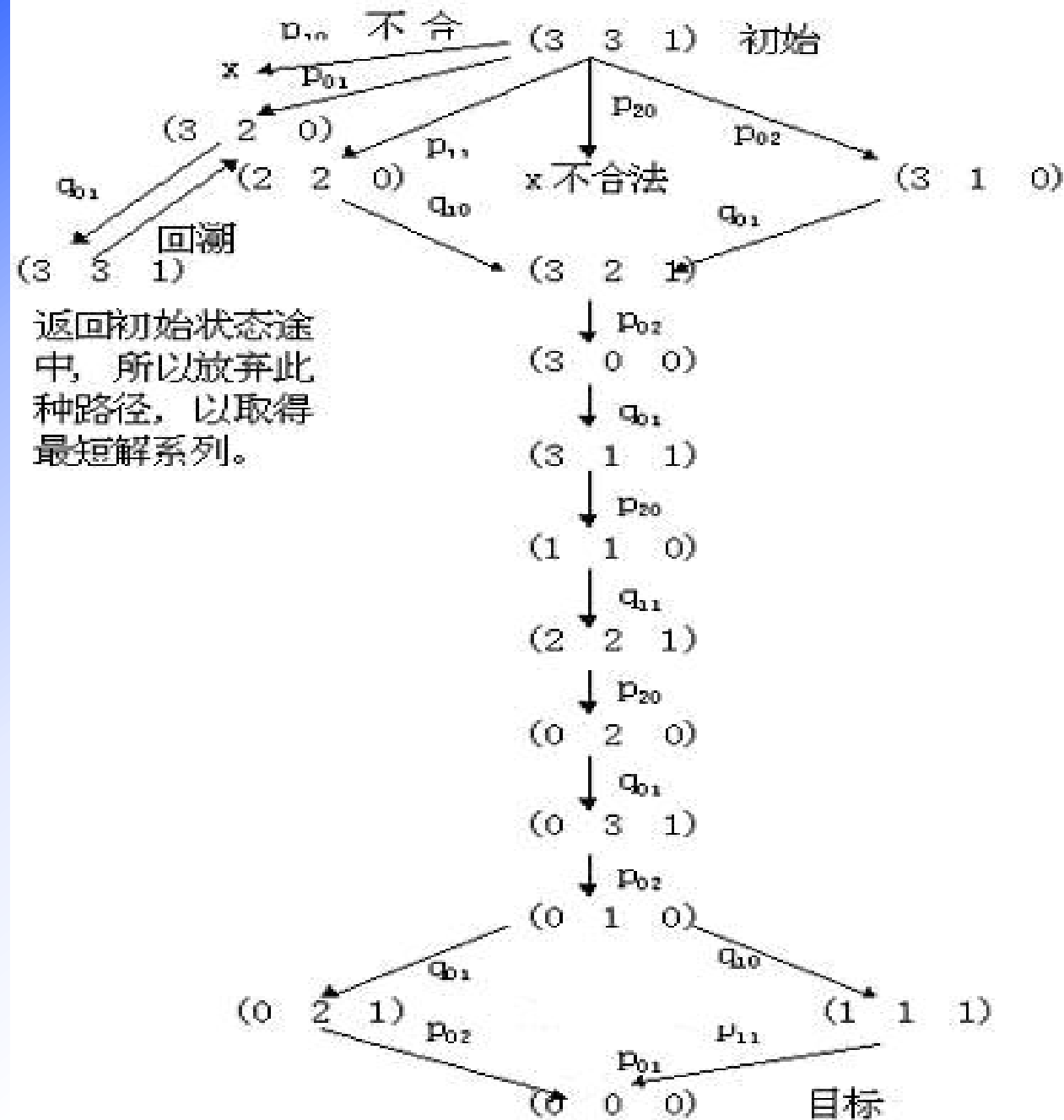
(3) 初始和目标状态: $(3, 3, 1)$ 和 $(0, 0, 0)$ 。

建立了产生式系统描述之后, 就可以通过控制策略, 对状态空间进行搜索, 求得一个摆渡操作序列, 使其实现目标状态。

状态空间图是一个有向图, 其节点可表示问题的各种状态(综合数据库), 节点之间的弧线代表一些操作(产生式规则), 它们可把一种状态导向另一种状态。这样建立起来的状态空间图, 描述了问题所有可能出现的状态及状态和操作之间的关系, 因而可以较直观地看出问题的解路径及其性质。

实际上只有问题空间规模较小的问题才可能作出状态空间图。由于每个摆渡操作都有对应的逆操作, 即 p_{mc} 对应 q_{mc} , 所以该图也可表示成具有双向弧的形式。





返回初始状态途中，
 所以放弃此种路径，
 以取得最短解系列。

M-C 问题状态空间图



从状态空间图看出解序列相当之多, 但最短解序列只有4个, 例如:

(p11、q10、p02、q01、p20、q11、p20、q01、p02、q01、p02)、

(p11、q10、p02、q01、p02、q11、p20、q01、p02、q10、p11)、

(p02、q01、p02、q01、p20、q11、p20、q01、p02、q01、p02)、

(p02、q01、p02、q01、p20、q11、p20、q01、p02、q10、p11),

若给定其中两个状态分别作为初始和目标状态, 就立即可找出对应的解序列来。
。在一般情况下, 求解过程就是对状态空间搜索出一条解路径的过程。

以上这个例子说明了建立产生式系统描述的过程, 这也就是所谓问题的表示。

对问题表示的好坏, 往往对求解过程的效率有很大影响。一种较好的表示法会简化状态空间和规则集表示。

其中的一条解路径为:

$(3\ 3\ 1) \rightarrow (3\ 1\ 0) \rightarrow (3\ 2\ 1) \rightarrow (3\ 0\ 0) \rightarrow (3\ 1\ 1) \rightarrow (1\ 1\ 0) \rightarrow (2\ 2\ 1) \rightarrow (0\ 2\ 0)$
 $\rightarrow (0\ 3\ 1) \rightarrow (0\ 1\ 0) \rightarrow (0\ 2\ 1) \rightarrow (0\ 0\ 0)$



用语句叙述的解路径(即过河方案)如下:

- (1) 初始状态: 3个传教士、3个野人和船均在左岸;
- (2) 2个野人由左岸过河到右岸;
- (3) 1个野人划船返回左岸;
- (4) 2个野人(包括返回的那个)由左岸过河到右岸;
- (5) 1个野人划船返回左岸;
- (6) 2个传教士由左岸过河到右岸;
- (7) 1个传教士和一个野人返回左岸;
- (8) 两个传教士(包括返回的那个)由左岸过河到右岸;
- (9) 1个野人返回左岸;
- (10) 2个野人由左岸过河到右岸;
- (11) 1野人返回左岸;
- (12) 2个野人由左岸过河到右岸, 至此, 传教士与野人全部过河, 此时3个传教士、3个野人和船全在右岸。

