


Chapter2

Problems, Problem Spaces and search

- 
- ❑ **To build a system to solve a particular problem, we need to do four things:**
 - **Define the problem precisely**
 - **Analyse the problem**
 - **Isolate and represent the task knowledge that is necessary to solve the problem**
 - **Choose the best problem-solving technique(s) and apply it(them) to the particular problem**



Defining the problem as a State Space Search

☐ Water jug Problem

- You are given two jugs, a 4-gallon one and a 3-gallon one. Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water. How can you get exactly 2 gallons of water into 4-gallon jug?
- ☐ State space for this problem can be described as the set of ordered pair of integers (x , y)
- ☐ $x = 0,1,2,3$ or 4:number of gallons of water in 4-gallon jug
- ☐ $y=0,1,2$ or 3:number of gallons of water in 3-gallon jug



Defining the problem as a State Space Search

Start state: (0 , 0)

Goal state:(2 , n)

- Assumptions:(Not mentioned in definition)**
 - We can pour water out of a jug onto ground**
 - We can pour water from one jug to another**

Production Rules

1	(x, y) if $x < 4$	$\rightarrow (4, y)$	Fill the 4-gallon jug
2	(x, y) if $y < 3$	$\rightarrow (x, 3)$	Fill the 3-gallon jug
3	(x, y) if $x > 0$	$\rightarrow (x - d, y)$	Pour some water out of the 4-gallon jug
4	(x, y) if $y > 0$	$\rightarrow (x, y - d)$	Pour some water out of the 3-gallon jug
5	(x, y) if $x > 0$	$\rightarrow (0, y)$	Empty the 4-gallon jug on the ground
6	(x, y) if $y > 0$	$\rightarrow (x, 0)$	Empty the 3-gallon jug on the ground
7	(x, y) if $x + y \geq 4$ and $y > 0$	$\rightarrow (4, y - (4 - x))$	Pour water from the 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full
8	(x, y) if $x + y \geq 3$ and $x > 0$	$\rightarrow (x - (3 - y), 3)$	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full
9	(x, y) if $x + y \leq 4$ and $y > 0$	$\rightarrow (x + y, 0)$	Pour all the water from the 3-gallon jug into the 4-gallon jug
10	(x, y) if $x + y \leq 3$ and $x > 0$	$\rightarrow (0, x + y)$	Pour all the water from the 4-gallon jug into the 3-gallon jug
11	$(0, 2)$	$\rightarrow (2, 0)$	Pour the 2 gallons from the 3-gallon jug into the 4-gallon jug
12	$(2, y)$	$\rightarrow (0, y)$	Empty the 2 gallons in the 4-gallon jug on the ground

Solution

Gallons in the 4-Gallon Jug	Gallons in the 3-Gallon Jug	Rule Applied
0	0	
0	3	2
3	0	9
3	3	2
4	2	7
0	2	5 or 12
2	0	9 or 11



Examples

- ❑ **Two jugs: 11 gallon, 4 gallon. 11 gallon must contain 1 gallon water**
- ❑ **Three jugs: 8 gallon, 5 gallon and 3 gallon. 8 gallon water must be divided equally among two jugs**



Solution of Example1

11-gallon jug	4-gallon jug
0	0
0	4
4	0
4	4
8	0
8	4
11	1
0	1
1	0



Solution of Example2

8-gallon jug	5-gallon jug	3-gallon jug
8	0	0
3	5	0
3	2	3
6	2	0
6	0	2
1	5	2
1	4	3
4	4	0

Informal Defⁿ → Formal Defⁿ

□ **Several issues that often arise in converting an informal problem statement into a formal problem description**

- 1. Role of conditions that occur in left side of the rules(i.e. Rule1)**
- 2. Rule 3 & 4 should they or should they not be included in the list of available operators?**
- 3. Special purpose rules that is to capture the special case knowledge that can be used at some stage(i.e. Rule 11,12)**
- 4. Level of precomputation in the rule (i.e. Rule 7,8)**

□ **Operationalization**

- A process in which program themselves produce formal descriptions from informal ones is known as operationalization**



Informal Defⁿ → Formal Defⁿ

□ To provide a formal description of a problem we must do following

- 1. Define a state space that contains all the possible configuration of the relevant objects**
- 2. Specify initial states**
- 3. Specify goal states**
- 4. Specify set of rules**

Need to consider

- Unstated assumption**
- How general should the rule be**
- Level of Precomputation**

The Missionaries and Cannibals Problem

- ❑ Three Missionaries and three cannibals find themselves on one side of a river. They have agreed that they would all like to get to the the other side. But the missionaries are not sure what else the cannibals have agreed to. So the missionaries want to manage the trip across the river in such a way that number of missionaries on either side of river is never less than the number of cannibals who are on the same side. The only boat available holds only two people at a time. How can everyone get across the river without the missionaries risking being eaten?
 - The state for this problem can be defined as $:\{(i,j)|i=0,1,2,3, j=0,1,2,3\}$ where i :Missionary and j :Cannibal
 - Initial state: (3,3) at bank1 (0,0) at bank2
 - Final state: (0,0) at bank1 (3,3) at bank2



Production Rules

1. (i, j) : Two Missionaries can go when $i-2 \geq j$ or $i=0$ on one side and $i+2 \geq j$ on the other side
2. (i, j) : Two Cannibals can go when $j-2 \leq i$ or $i=0$ on one side and $j+2 \leq i$ or $i=0$ on the other side
3. (i, j) : One Missionary and one Cannibal can go when $i-1 \geq j-1$ or $i=0$ on one side and $i+1 \geq j+1$ on the other side
4. (i, j) : One Missionary can go when $i-1 \geq j$ or $i=0$ on one side and $i+1 \geq j$ on the other side
5. (i, j) : One cannibal can go when $i \geq j-1$ or $i=0$ on one side and $i \geq j+1$ or $i=0$ on the other side



Problem Solution

Bank1		Bank2
(3,3)		(0,0)
(3,1)	→	(0,2)
(3,2)	←	(0,1)
(3,0)	→	(0,3)
(3,1)	←	(0,2)
(1,1)	→	(2,2)
(2,2)	←	(1,1)
(0,2)	→	(3,1)
(0,3)	←	(3,0)
(0,1)	→	(3,2)
(0,2)	←	(3,1)
(0,0)	→	(3,3)

Problem, State Space Representation

State Space Representation for Chess

Example: Playing Chess

- To build a program that could “play chess”, we could first have to specify the starting position of the chess board, the rules that define the legal moves, and the board positions that represent a win for one side or the other.

Playing chess

- The starting position can be described as an 8by 8 array where each position contains a symbol for appropriate piece.
- We can define as our goal the check mate position.
- The legal moves provide the way of getting from initial state to a goal state.
- They can be described easily as a set of rules consisting of two parts:
 - A left side that serves as a pattern to be matched against the current board position.
 - And a right side that describes the change to be made to reflect the move
- However, this approach leads to large number of rules 10^{120} board positions !!
- Using so many rules poses problems such as:
 - No person could ever supply a complete set of such rules.
 - No program could easily handle all those rules. Just storing so many rules poses serious difficulties.

Defining chess problem as State Space search

- We need to write the rules describing the legal moves in as general a way as possible.
- For example:
 - White pawn at Square(file e, rank 2) AND Square(File e, rank 3) is empty AND Square(file e, rank 4) is empty, then move the pawn from Square(file e, rank 2) to Square(file e, rank 4).
- In general, the more succinctly we can describe the rules we need, the less work we will have to do to provide them and more efficient the program.

Travelling Salesman Problem

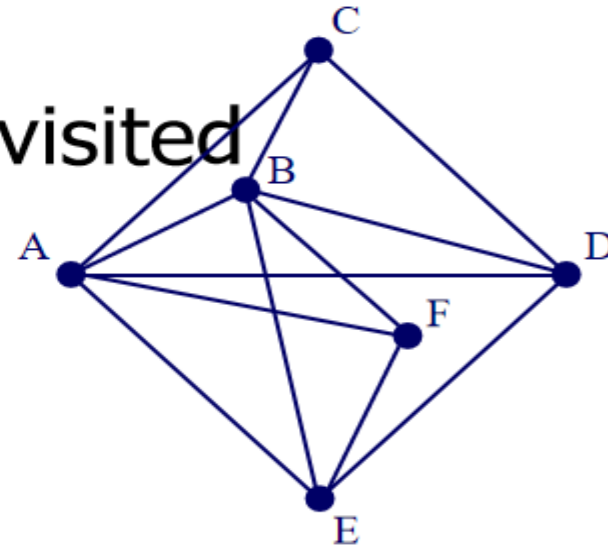
- a common variation of state-space problem require finding not just any path but one of minimum cost between an initial node and a goal node. e.g. Traveling Salesman Problem

- State: sequence of cities visited

- $S_0 = A$

- $S_G =$ a complete tour

$$\{a, c, d\} \Rightarrow \{(a, c, d, x) \mid x \notin \{a, c, d\}\}$$



	A	B	C	D
A	—	4	6	10
B		—	7	10
C			—	5
D				—

Mileage chart

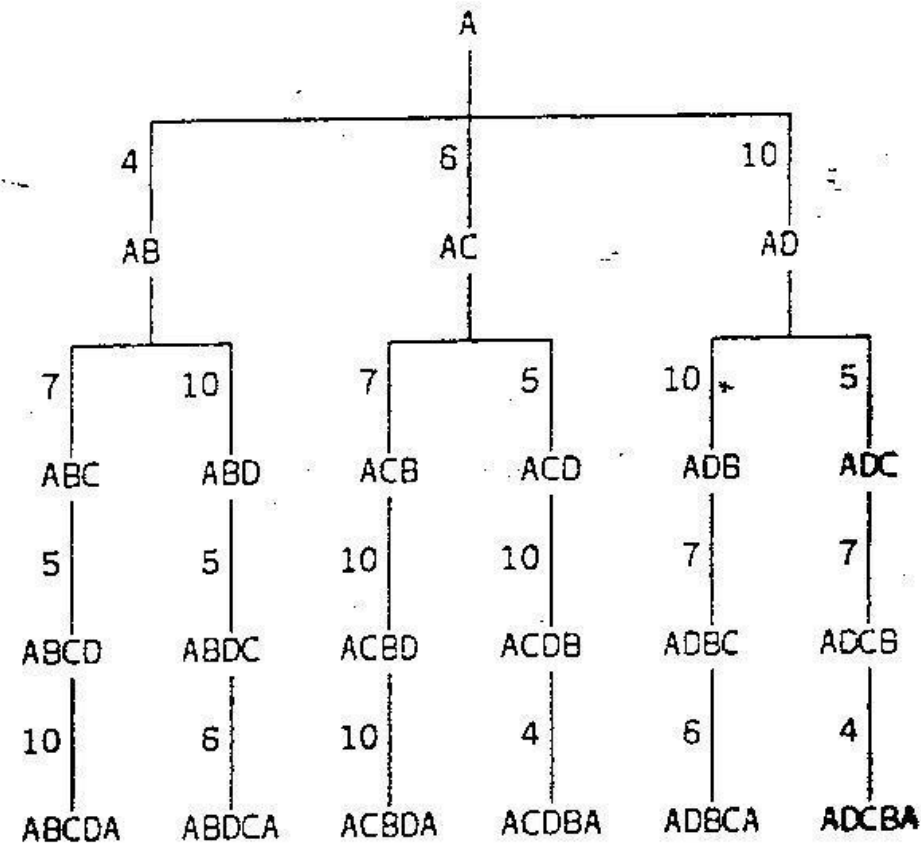


Figure B1-5. The state-space graph for a traveling-salesman problem.

Example: Eight puzzle (8-Puzzle)

- The 8-puzzle is a 3×3 array containing eight square pieces, numbered 1 through 8, and one empty space. A piece can be moved horizontally or vertically into the empty space, in effect exchanging the positions of the piece and the empty space. There are four possible moves, UP (move the blank space up), DOWN, LEFT and RIGHT. The aim of the game is to make a sequence of moves that will convert the board from the start state into the goal state:

2	3	4
8	6	2
7		5

Initial State

1	2	3
8		4
7	6	5

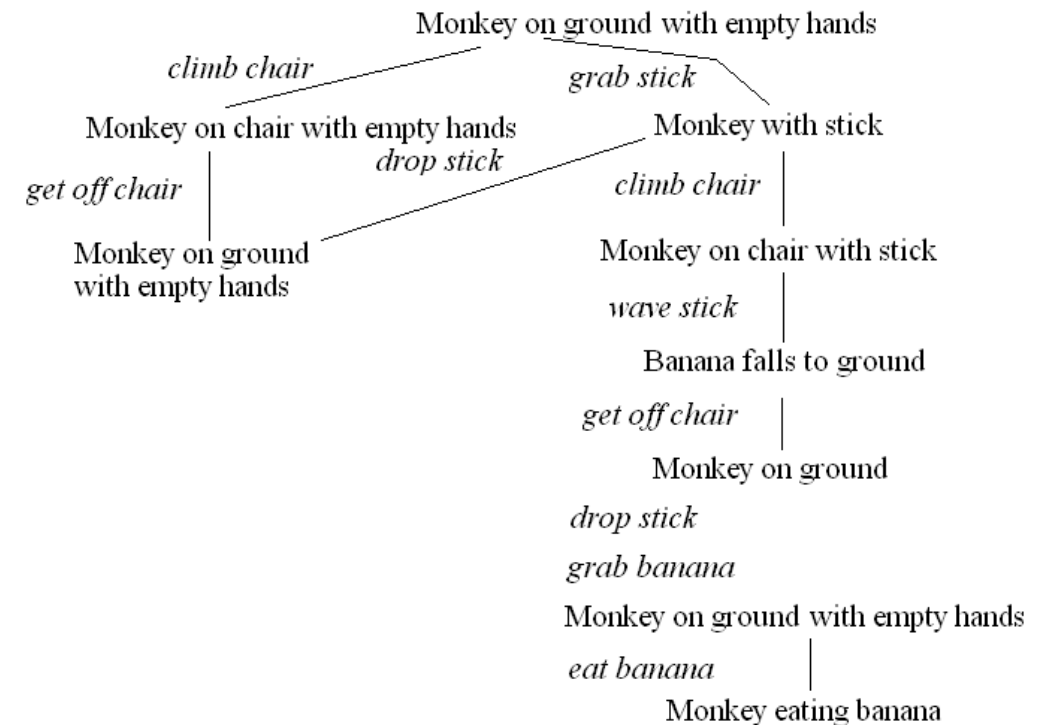
Goal State

This example can be solved by the operator sequence UP, RIGHT, UP, LEFT, DOWN.

The Monkey & Bananas Problem

- A monkey is in a cage and bananas are suspended from the ceiling, the monkey wants to eat a banana but cannot reach them
 - in the room are a chair and a stick
 - if the monkey stands on the chair and waves the stick, he can knock a banana down to eat it
 - what are the actions the monkey should take?

- Initial state:
 - monkey on ground with empty hand
 - bananas suspended
- Goal state:
 - monkey eating
- Actions:
 - climb chair/get off
 - grab X
 - wave X
 - eat X



Production Systems

- The core of many intelligent systems :

Search

Production System consists of :

- So now we need to structure the AI programs that we create in such a way that it facilitates describing and performing the search process.
 - A set of rules, consisting of left side that determines the applicability of the rule and a right side that describes the operation to be performed if that rule is applied.
 - One or more knowledge/databases
 - A control strategy which specifies the order in which the rules will be compared to the database and a way of resolving the conflicts which may arise when several rules match at once
 - And a rule applier

In order to solve a problem -

- We must first reduce it to one for which a precise statement can be given.
- This can be done by defining the problem's state space (start and goal state) and a set of operators for moving that space.
- The problem can then be solved by searching for a path through the space from an initial state to a goal state.
- Modeling this whole process is called as production system.

Control Strategies

- Water Jug Problem
- 8 Puzzle Problem
- Travelling Salesman Problem
 - One important Question that arises while solving these AI problems is :
 - How to decide which rule to apply next during the process of searching for a solution to a problem.

1	(x, y) if $x < 4$	$\rightarrow (4, y)$	Fill the 4-gallon jug
2	(x, y) if $y < 3$	$\rightarrow (x, 3)$	Fill the 3-gallon jug
3	(x, y) if $x > 0$	$\rightarrow (x - d, y)$	Pour some water out of the 4-gallon jug
4	(x, y) if $y > 0$	$\rightarrow (x, y - d)$	Pour some water out of the 3-gallon jug
5	(x, y) if $x > 0$	$\rightarrow (0, y)$	Empty the 4-gallon jug on the ground
6	(x, y) if $y > 0$	$\rightarrow (x, 0)$	Empty the 3-gallon jug on the ground
7	(x, y) if $x + y \geq 4$ and $y > 0$	$\rightarrow (4, y - (4 - x))$	Pour water from the 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full
8	(x, y) if $x + y \geq 3$ and $x > 0$	$\rightarrow (x - (3 - y), 3)$	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full
9	(x, y) if $x + y \leq 4$ and $y > 0$	$\rightarrow (x + y, 0)$	Pour all the water from the 3-gallon jug into the 4-gallon jug
10	(x, y) if $x + y \leq 3$ and $x > 0$	$\rightarrow (0, x + y)$	Pour all the water from the 4-gallon jug into the 3-gallon jug
11	$(0, 2)$	$\rightarrow (2, 0)$	Pour the 2 gallons from the 3-gallon jug into the 4-gallon jug
12	$(2, y)$	$\rightarrow (0, y)$	Empty the 2 gallons in the 4-gallon jug on the ground

\Rightarrow Suppose consider.

\rightarrow The current state in water Jug is

$\boxed{(0, 3)}$

\rightarrow Now the possible applicable rule in

this case is,

(Rule 6)

\rightarrow Here, $y > 0$ \rightarrow I can do is $\rightarrow (0, 0)$
4 litre

or Pour some to Jug,

new state $(2, 1)$, (Rule 4)

or, Rule 9 where $x+y \leq 4$ and
 $y > 0$

which says $\rightarrow (3, 0) \rightarrow$ so makes
 $(x+y, 0)$

→ How to decide which Rule to
apply next?

⇒ Sometimes it may also happen that
we keep applying the first applicable
one while starting from top.

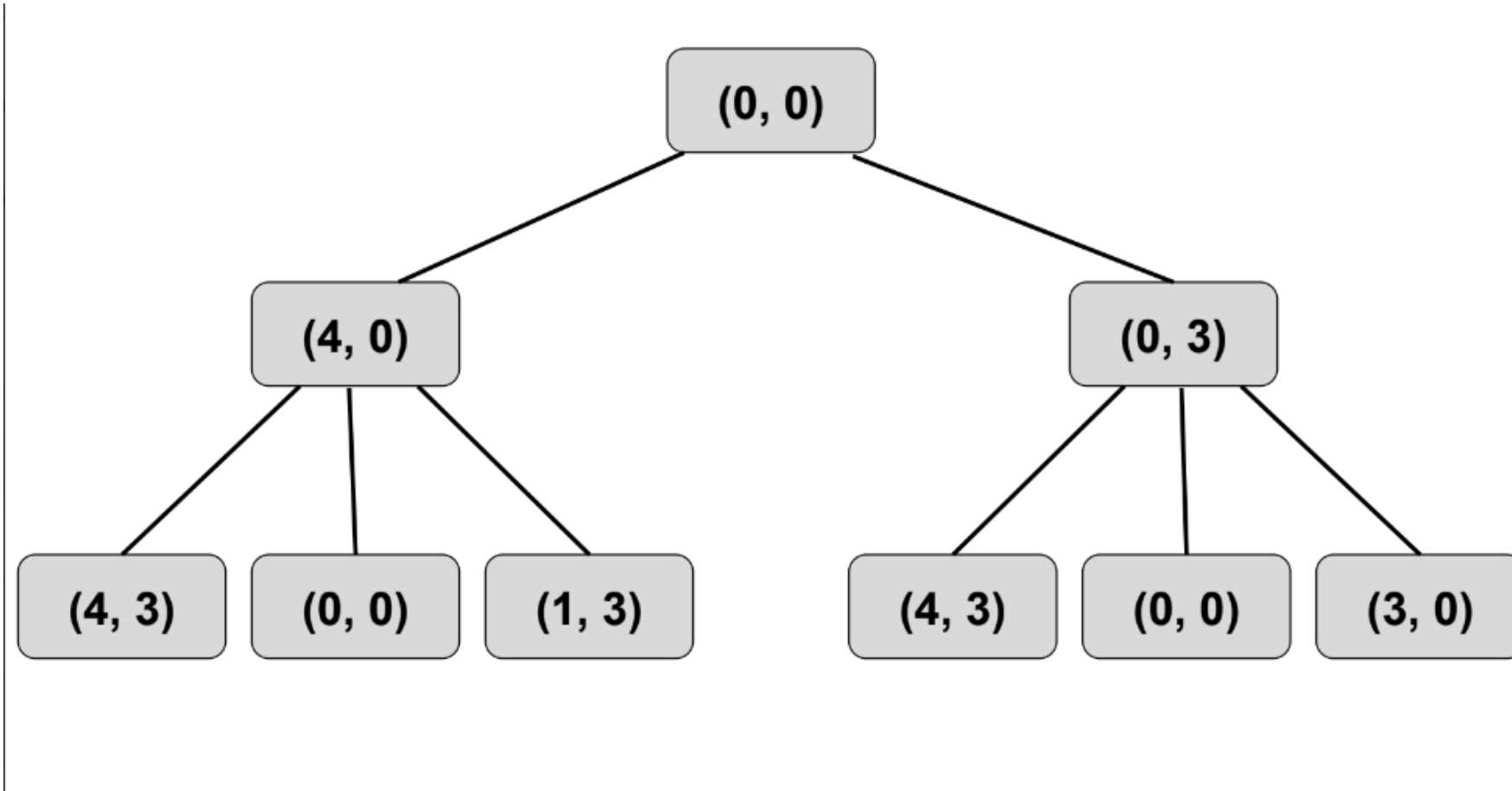
⇒ This control strategy does not
cause motion.

→ Not systematic
↳ one solution,
Choose rule
randomly.

Control Strategies

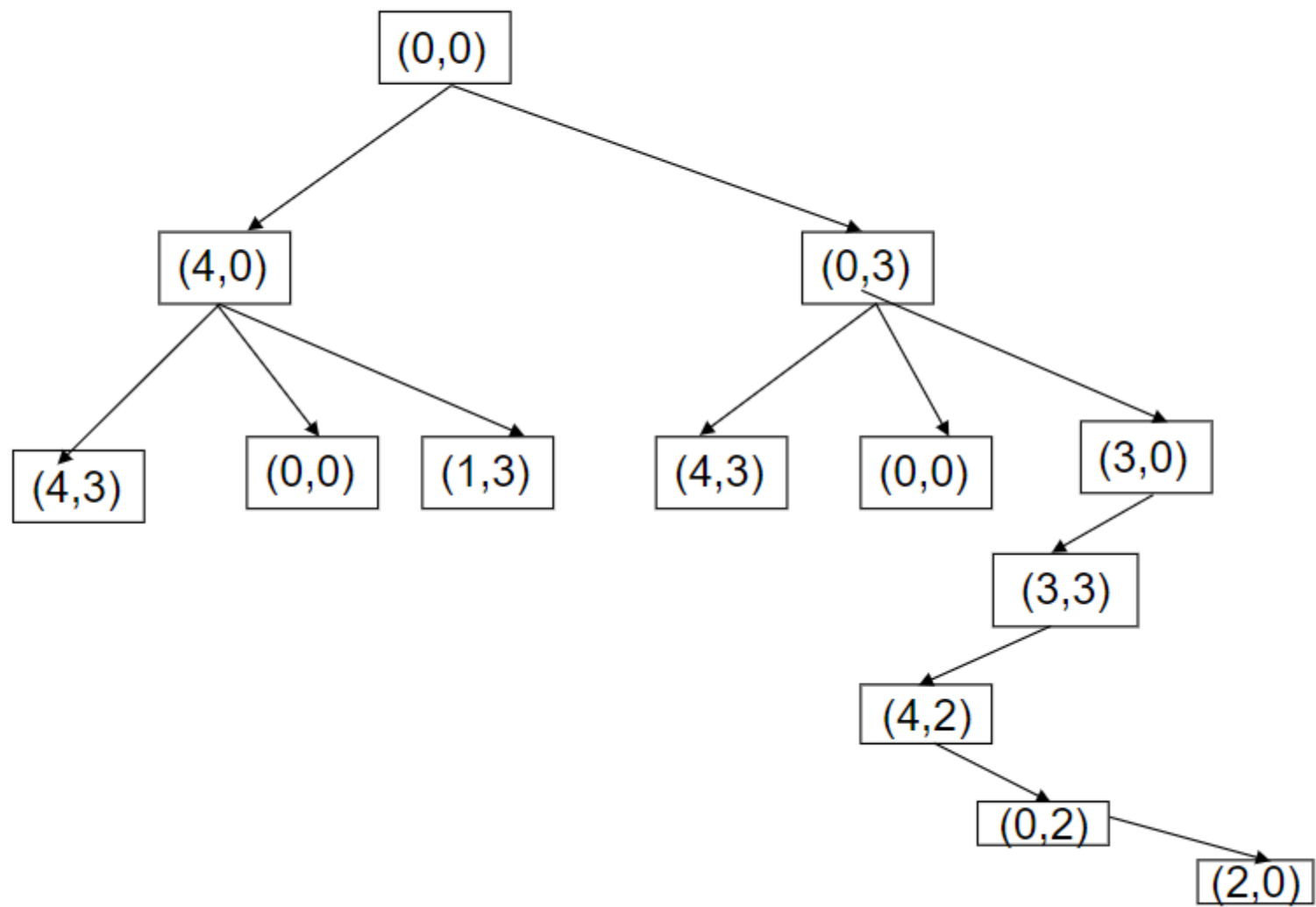
- How to decide which rule to apply next during the process of searching for a solution to a problem?
- The two requirements of good control strategy are that
 - it should cause motion (causes local motion)
 - It should be systematic (causes global motion)e.g. Breadth First Search

Systematic Control Strategy

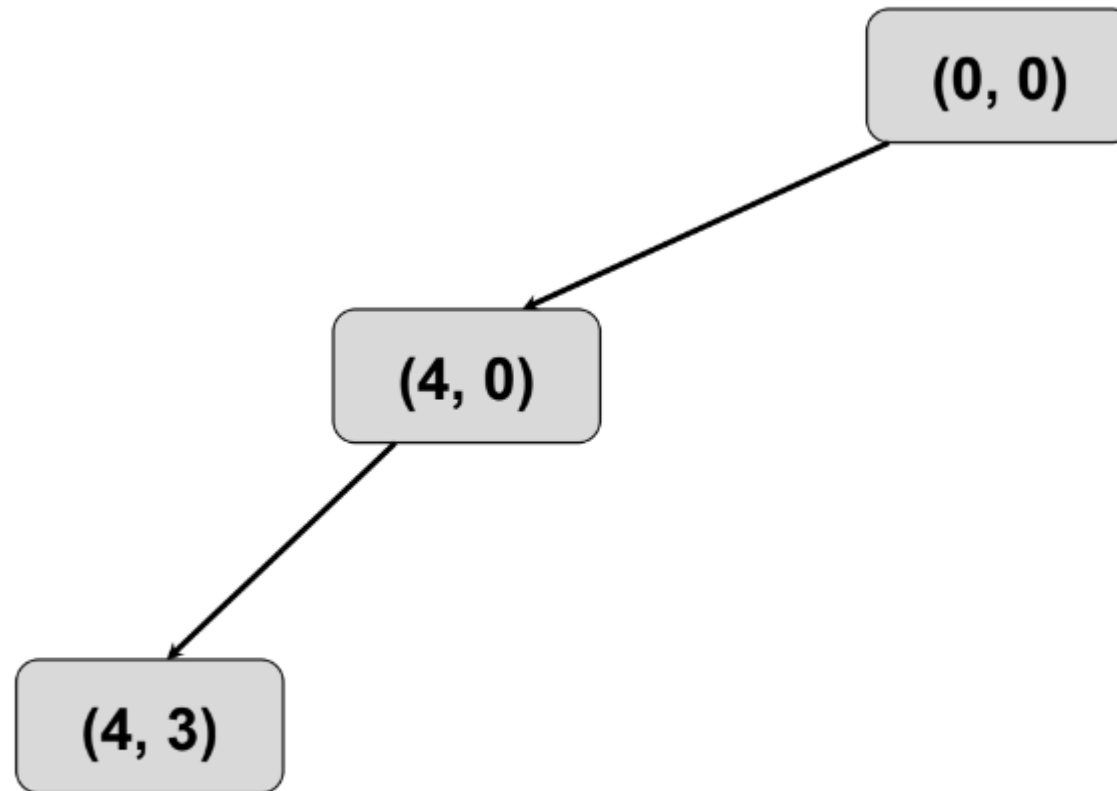


BFS

- Algorithm:
 1. Create a variable called NODE-LIST and set it to initial state
 2. Until a goal state is found or NODE-LIST is empty do
 - a. Remove the first element from NODE-LIST and call it E. If NODE-LIST was empty, quit
 - b. For each way that each rule can match the state described in E do:
 - i. Apply the rule to generate a new state
 - ii. If the new state is a goal state, quit and return this state
 - iii. Otherwise, add the new state to the end of NODE-LIST



Depth First Search



1. If the initial state is a goal state, quit and return success
2. Otherwise, do the following until success or failure is signaled:
 - a. Generate a successor, E, of initial state. If there are no more successors, signal failure.
 - b. Call Depth-First Search, with E as the initial state
 - c. If success is returned, signal success. Otherwise continue in this loop.

Backtracking

- Backtracking occurs when -
 - If it reaches dead-end
 - produces a previous state or
 - become longer than some previous limit.
- Chronological Backtracking
 - The order in which steps are undone depends only on the temporal sequence in which steps were initially made.
- Specifically most recent step is always the first to be undone.

Advantages of Depth-First Search

- DFS requires less memory since only the nodes on the current path are stored.
- By chance, DFS may find a solution without examining much of the search space at all.

Advantages of BFS

- BFS will not get trapped exploring a blind alley.
- If there is a solution, BFS is guaranteed to find it.
- If there are multiple solutions, then a minimal solution will be found.

Travelling Salesman Problem

- A simple motion causing and systematic control structure could solve this problem.
- Simply explore all possible paths in the tree and return the shortest path.
- If there are N cities, then number of different paths among them is $1.2....(N-1)$ or $(N-1)!$
- The total time required to perform this search is proportional to $N!$
- This phenomenon is called Combinatorial explosion.

Branch and Bound

- Begin generating complete paths, keeping track of the shortest path found so far.
- Give up exploring any path as soon as its partial length becomes greater than the shortest path found so far.
- Using this algorithm, we are guaranteed to find the shortest path.
- It requires exponential time.
- The time it saves depends on the order in which paths are explored.

Heuristic Search

- A Heuristic is a technique that improves the efficiency of a search process, possibly by sacrificing claims of completeness.
- On the average they improve the quality of the paths that are explored.
- Using Heuristics, we can hope to get good (though possibly non-optimal) solutions to hard problems such as a TSP in non exponential time.
- There are good general purpose heuristics that are useful in a wide variety of problem domains.
- Special purpose heuristics exploit domain specific knowledge

Nearest Neighbor Heuristic

- It works by selecting locally superior alternative at each step.
- Applying to TSP:
 1. Arbitrarily select a starting city
 2. To select the next city, look at all cities not yet visited and select the one closest to the current city. Go to next step.
 3. Repeat step 2 until all cities have been visited.
- This procedure executes in time proportional to N^2
- This provides reassurance that one is not paying too high a price in accuracy for speed.

Why do we use Heuristics ?

1. Rarely do we actually need the optimal solution
2. Although the approximation produced by heuristics may not be very good in the worst case, worst case rarely arise in the real world
3. Trying to understand why heuristic works or why it doesn't work often leads to a deeper understanding of the problem

- ❑ The purpose of heuristic function is to guide the search process in the most profitable direction by suggesting which path to follow first when more than one is available**
- ❑ There is a trade-off between the cost of evaluating a heuristic function and the saving in search time that the function provides**

Heuristic Function

- This is a function that maps from problem state descriptions to measures of desirability, usually represented as numbers.
 - Which aspects of the problem state are considered,
 - how those aspects are evaluated, and
 - the weights given to individual aspects are chosen in such a way that
- the value of the heuristic function at a given node in the search process gives as good an estimate as possible of whether that node is on the desired path to a solution.
- Well designed heuristic functions can play an important part in efficiently guiding a search process towards a solution.

- Studying for exam
- Tour Guide

Heuristic for 8 Puzzle Problem

5		8
4	2	1
7	3	6

Start

1	2	3
4	5	6
7	8	

Goal

Heuristic1 = number of misplaced numbered tiles

5		8
4	2	1
7	3	6

H=6

5	8	
4	2	1
7	3	6

H=6

	5	8
4	2	1
7	3	6

H=6

5	2	8
4		1
7	3	6

H=5

5		8
4	2	1
7	3	6

Start

1	2	3
4	5	6
7	8	

Goal

**Heuristic2 = sum of the (Manhattan) distance of every
numbered tile to its goal position**

$$= 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1$$

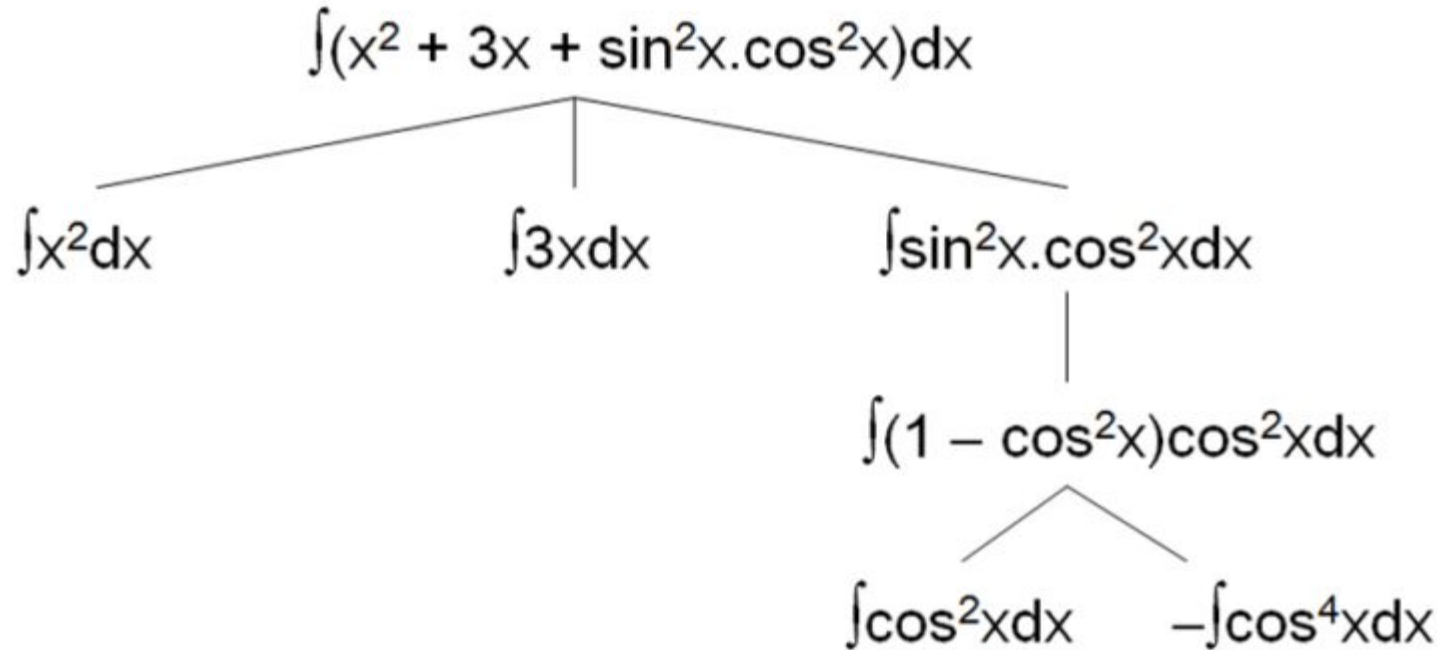
$$= 13$$

Problem Characteristics

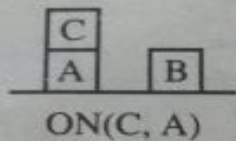
Why ?

- Analyzing the problem across several key dimensions.
- In order to choose the most appropriate method to solve a problem, the nature of problem needs to be analyzed.

Is the problem decomposable ?



Start:



Goal:

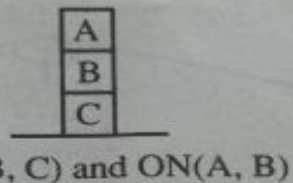
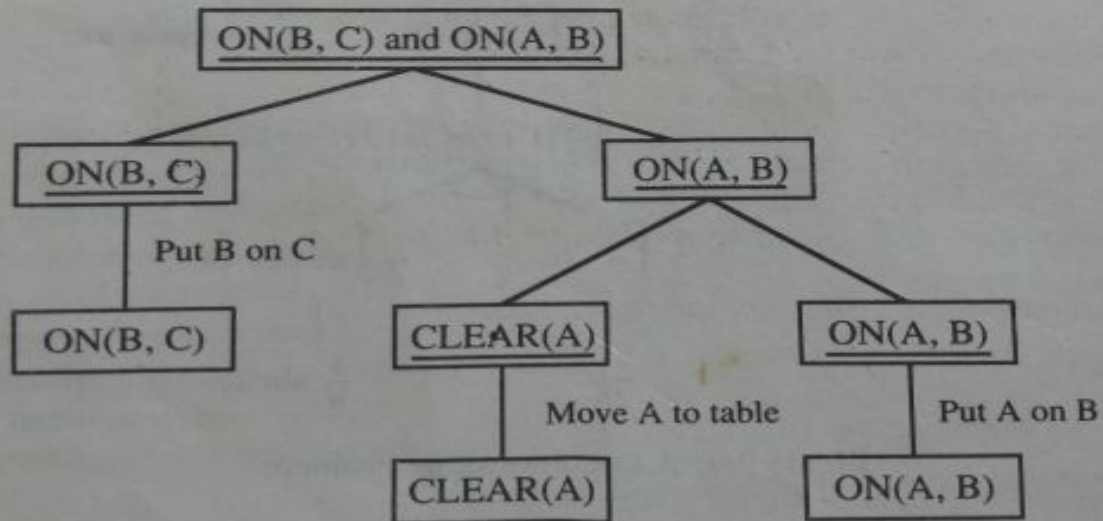


Figure 2.10: A Simple Blocks World Problem



Assume that the following operators are available

1. CLEAR(x)[block x has nothing on it]ON(x, Table) [pick up x and put it on the table]
2. CLEAR(x) and CLEAR(y) ON(x ,y) [put x on y]

Can solution steps be ignored or undone?

Ignorable: In which solution steps can be ignored

e.g . Theorem proving

Recoverable: In which solution steps can be undone

e.g . 8-puzzle

Irrecoverable: In which solution steps cannot be undone

e.g . Chess

Cont.

Recoverability of a problem plays an important role in determining the complexity of control structure necessary for the problem solution.

Ignorable: Simple control structure that never backtracks

Recoverable: Slightly more complicated control structure that allows backtracking in case of mistakes e.g. stack is useful

Irrecoverable: Will need to have control structure that expends deal of effort making each decision since the decision must be final

Is the universe predictable?

- Certain Outcome (ex: 8-puzzle)
 - One way of describing planning is that it is problem solving without feedback from the environment
 - For solving certain outcome problems, open loop approach will work fine since the result of action can predicted perfectly
 - Planning can be used to generate a sequence of operators that is guaranteed to lead to a solution

- Uncertain Outcome (ex: Bridge)
 - Planning can at best generate a sequence of operators that has a good probability of leading to a solution
 - We need to allow for a process of plan revision to take place as the plan is carried out and the necessary feedback is provided
 - Providing no guarantee of an actual solution
 - Planning is very expensive since the number of solution paths that need to be explored increases exponentially with the number of points at which the outcome cannot be predicted

Irrecoverable Uncertain Problems

1. Playing Bridge
2. Controlling a robot arm.

Is Good solution Absolute or Relative?

Consider the problem of answering questions based on a database of simple facts, such as the following: Is Marcus Alive?

1. Marcus was a man.
2. Marcus was a Pompeian.
3. Marcus was born in 40 A.D.
4. All men are mortal.
5. All Pompeians died when the volcano erupted in 79 A.D.
6. No mortal lives longer than 150 years.
7. It is now 1991 A.D.

Justification

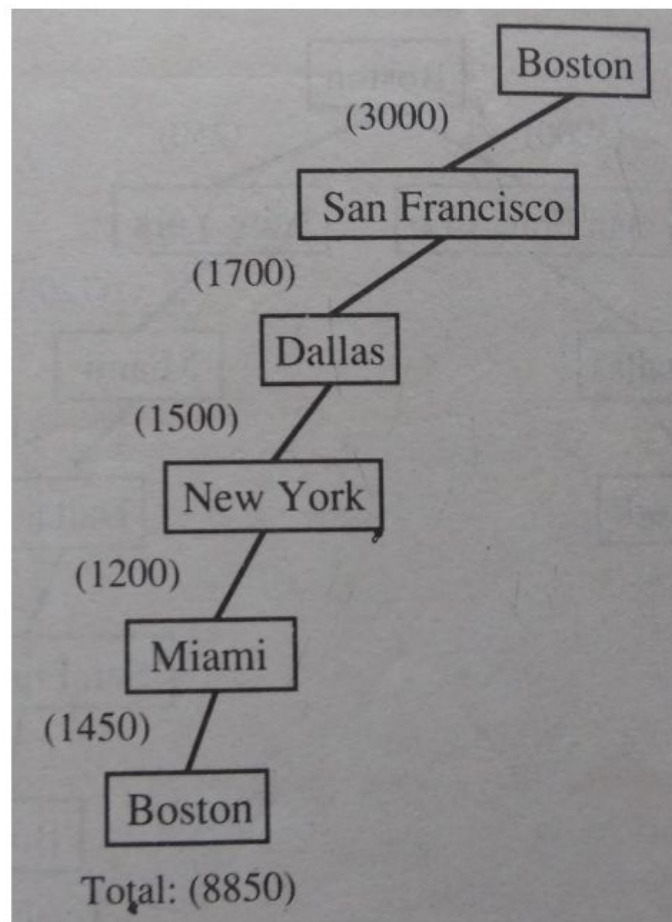
- | | | |
|-----|--|---------|
| 1. | Marcus was a man. | axiom 1 |
| 4. | All men are mortal. | axiom 4 |
| 8. | Marcus is mortal. | 1, 4 |
| 3. | Marcus was born in 40 A.D. | axiom 3 |
| 7. | It is now 1991 A.D. | axiom 7 |
| 9. | Marcus' age is 1951 years. | 3, 7 |
| 6. | No mortal lives longer than 150 years. | axiom 6 |
| 10. | Marcus is dead. | 8, 6, 9 |

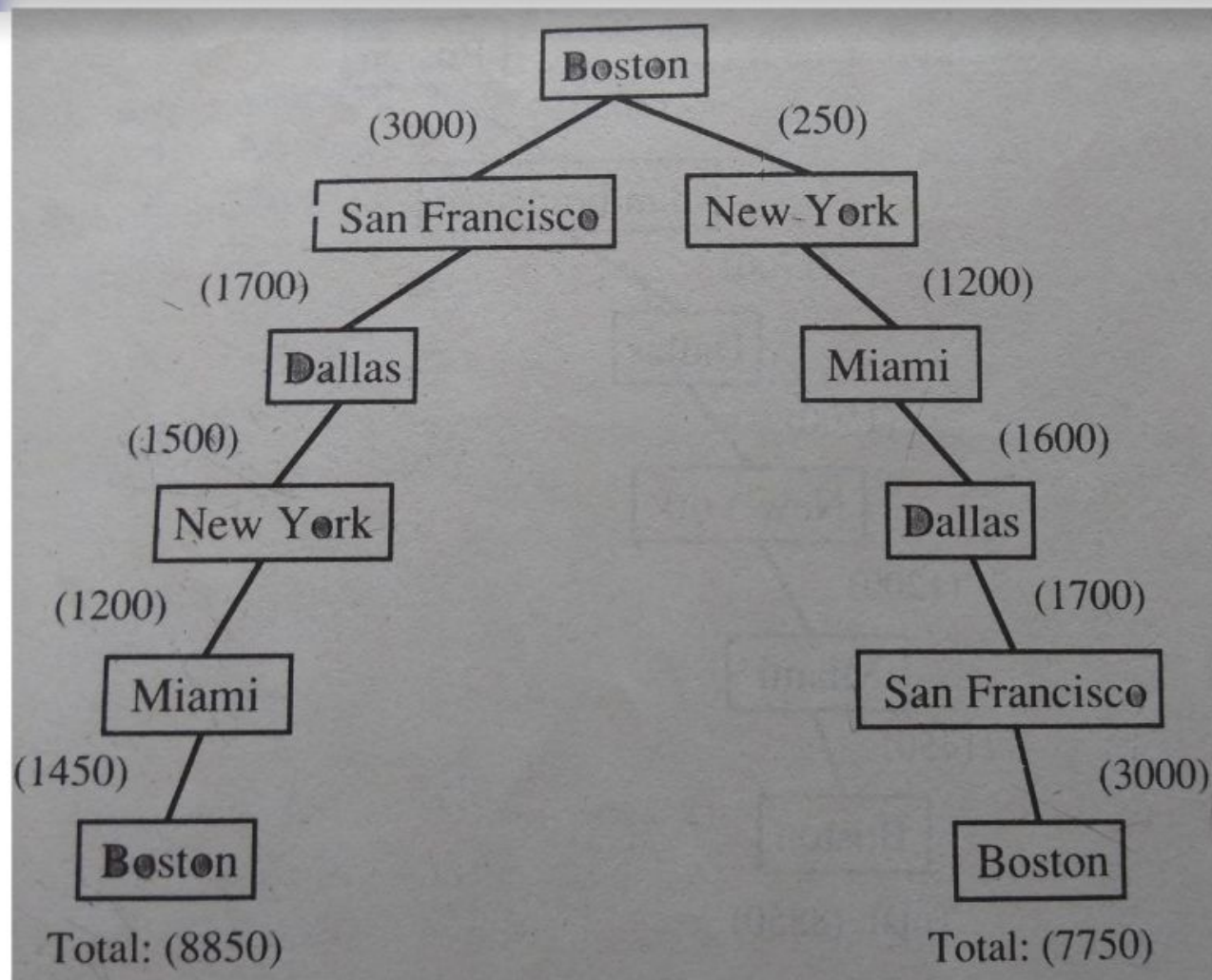
OR

- | | | |
|-----|-------------------------------|---------|
| 7. | It is now 1991 A.D. | axiom 7 |
| 5. | All Pompeians died in 79 A.D. | axiom 5 |
| 11. | All Pompeians are dead now. | 7, 5 |
| 2. | Marcus was a Pompeian. | axiom 2 |
| 12. | Marcus is dead. | 11, 2 |

Consider the Travelling salesman problem. Our goal is to find the shortest route that visits each city exactly once

	Boston	New York	Miami	Dallas	S.F.
Boston		250	1450	1700	3000
New York	250		1200	1500	2900
Miami	1450	1200		1600	3300
Dallas	1700	1500	1600		1700
S.F.	3000	2900	3300	1700	





Any –path problem

- Any-path problems can often be solved in a reasonable amount of time by using heuristics that suggest good paths to explore

Best-path problem

- In Best-path problem, no heuristic that could possibly miss the best solution can be used. So a much more exhaustive search will be performed

Is the solution a state or a path?

Consider a problem of **finding a consistent interpretation for the sentence**

“The bank president ate a dish of pasta salad with fork”

We need only final interpretation, not the record of processing by which the interpretation was found

Consider the **Water jug problem**

A statement of a solution to this problem must be a sequence of operations that produces the final state
Here, it is not sufficient to report that we have solved the problem and that final state is (2,0). For this kind of problem, What we really must report is not the final state but the path that we found to that state

What is the role of knowledge?

Examples:

Playing chess: Knowledge required to constrain the search for a solution

Newspaper story understanding: Lot of knowledge is required even to be able to recognize a solution

❑ Consider the problem of scanning daily newspapers

“ To decide which are supporting the Democrats and which are supporting Republicans in some upcoming election”

We need lot of knowledge to answer such question as:

- The names of the candidates in each party**
- The fact that if the major thing you want to see done is have taxes lowered, you are probably supporting the Republicans**
- The fact that if the major thing you want to see done is improved education for minority students, you are probably supporting the Democrats**
- And so on...**

Does the task require interaction with a person?

Sometimes it is useful to program computers to solve problems in ways that the majority of people would not be able to understand. This is fine if the level of the interaction between the computer and its human users is **problem-in solution out**

But increasingly we are building programs that require intermediate interaction with people, both to provide additional input to the program and to provide additional reassurance to the user

□ Types of problems:

- **Solitary:** In which the computer is given a problem description and produces an answer with no intermediate communication and with no demand for an explanation of the reasoning process
- **Conversational:** In which there is intermediate communication between a person and the computer, either to provide additional assistance to the computer or to provide additional information to the user or both

Example: Theorem Proving

Medical Diagnosis

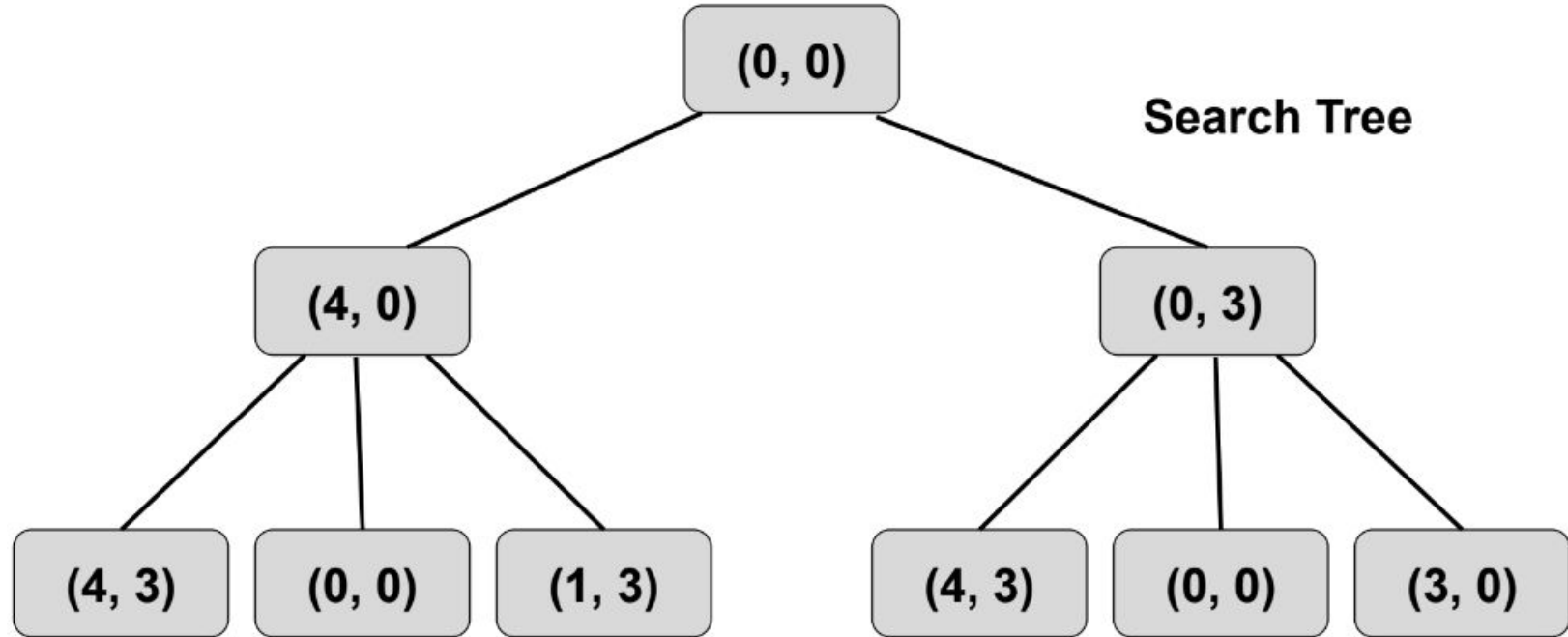
- Production systems are a good way to describe the operations that can be performed in a search for a solution to a problem.
1. Can production systems, like problems, be described by a set of characteristics that shed some light on how they easily be implemented?
 2. If so, what relationships are there b/w problem types and the types of production systems best suited to solve the problem.

- A ***monotonic production system*** is a system in which the application of rule never prevents the later application of another rule that could also have been applied at the time that the first rule was selected.
- A ***nonmonotonic production system*** is one in which this is not true.
- A ***partially commutative production system*** is a system in with the property that if the application of particular sequence of rules transforms state x into state y, then any permutation of those rules that is allowable also transform state x in to state y.
- A ***commutative production system*** is a production system that is both monotonic and partially commutative.

	Monotonic	Non-Monotonic
Partially Commutative	Theorem Proving	Robot Navigation
Non-Partially Commutative	Chemical Synthesis	Bridge , Chess

Issues in the design of search programs

Every search process can be viewed as a traversal of tree structure



- Explicit search tree: First create search tree explicitly and then exploration
- Implicit search tree: Generate only part that needs to explore

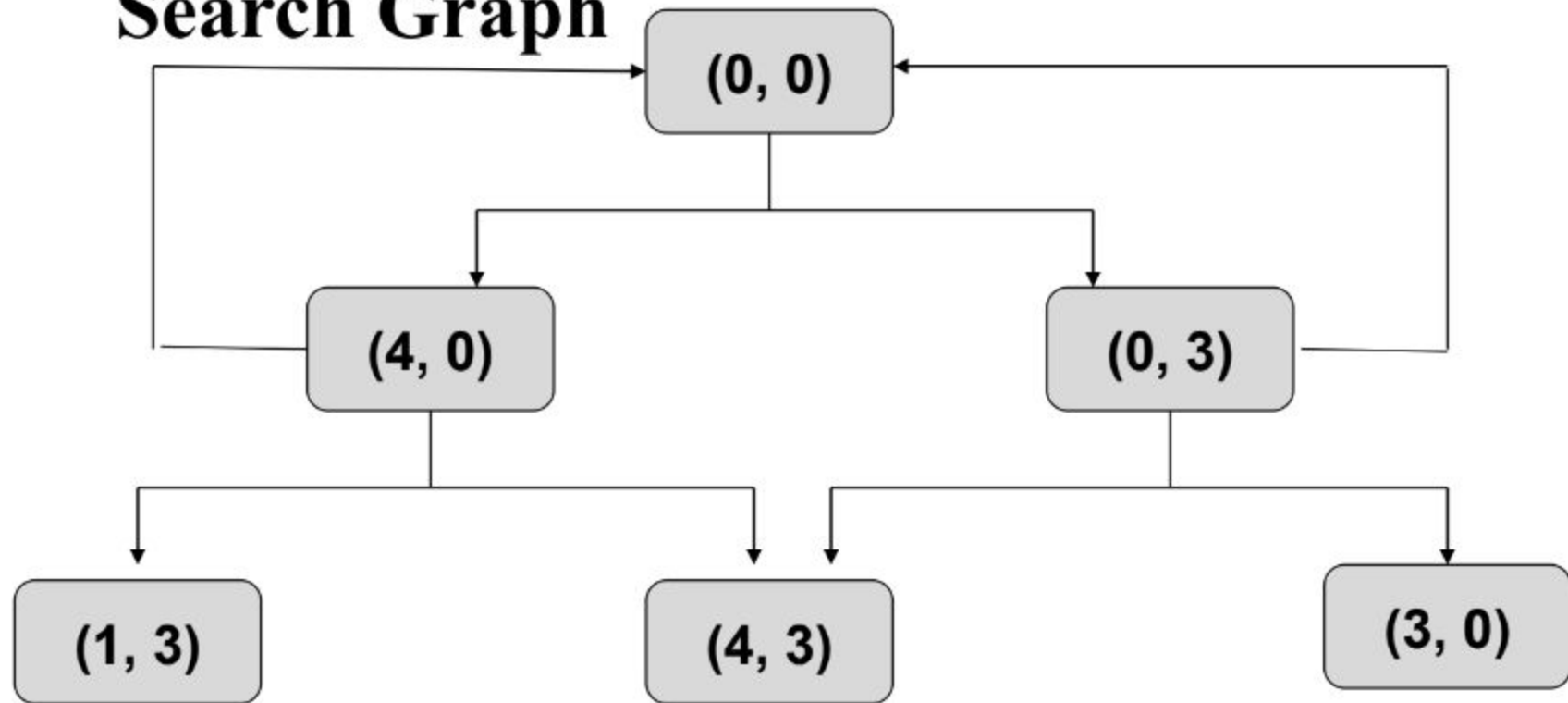
□ Issues in search program

- 1. The direction in which to conduct search (Forward VS Backward reasoning)**
- 2. How to select applicable rules(matching): Need an efficient procedure for matching rules against states**
- 3. How to represent each node of the search process**

Search Tree

- ❑ Same node generates twice (i.e. $(4,3)$)
- ❑ Start state is obtained again that means exploration of path is not leading us to a solution (i.e. $(0,0)$)
- ❑ So, eliminate them and try other paths

Search Graph



Algorithm: Check Duplicate Nodes

- 1. Examine the set of nodes that have been created so far to see if the new node already exists**
- 2. If it does not, simply add it to the graph just as for a tree**
- 3. If it does already exist, then do the following:**
 - a. Set the node that is being expanded to point to the already existing node corresponding to its successor rather than to the new one. The new one can simply be thrown away**
 - b. If you are keeping track of the best(shortest or otherwise least-cost) path to each node, then check to see if the new path is better or worse than the old one. If worse, do nothing. If better, record the new path as the correct path to use to get to the node and propagate the corresponding change in cost down through successor nodes as necessary**

Problem in Search Graphs

Additional bookkeeping is necessary to test duplicate nodes

Cycle may be introduced into the graph search