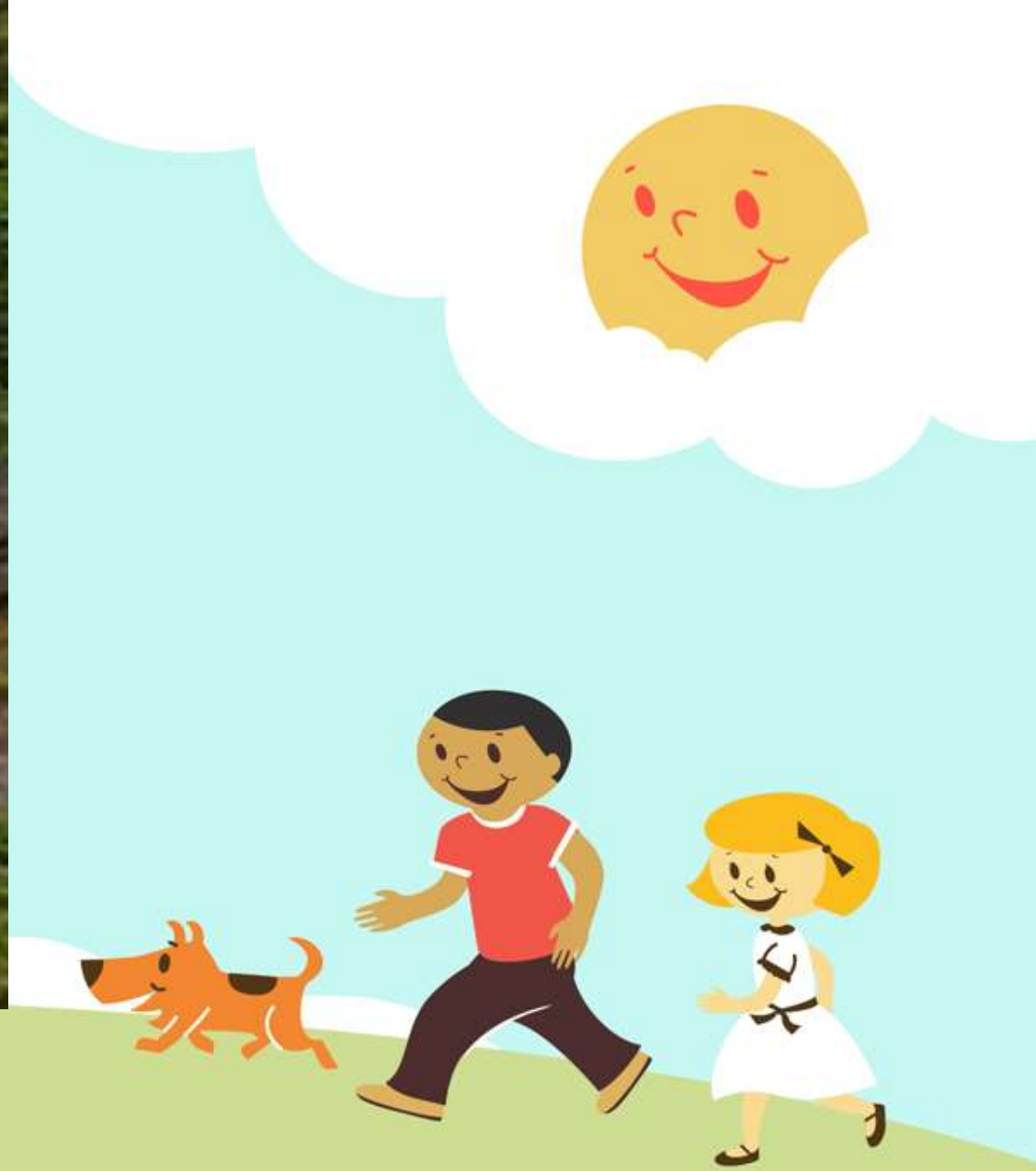


Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down as far I could
To where it bent in the undergrowth;
Then took the other….

Structures and Strategies for State Space Search

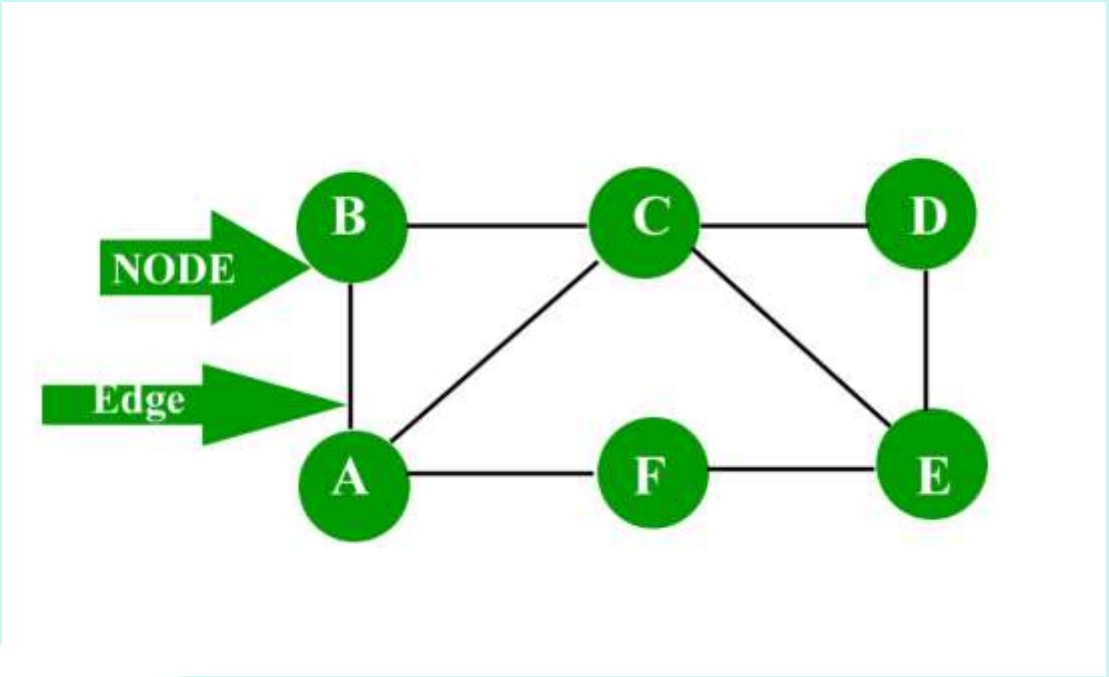
Questions to be answered?

- Is the problem solver guaranteed a solution?
- Will the search terminate or get caught in an infinite loop?
- Will the solution be optimal?
- What are the time and memory usage complexity?

Konigsberg Bridge Problem

Graph Theory

Structures for State Space Search



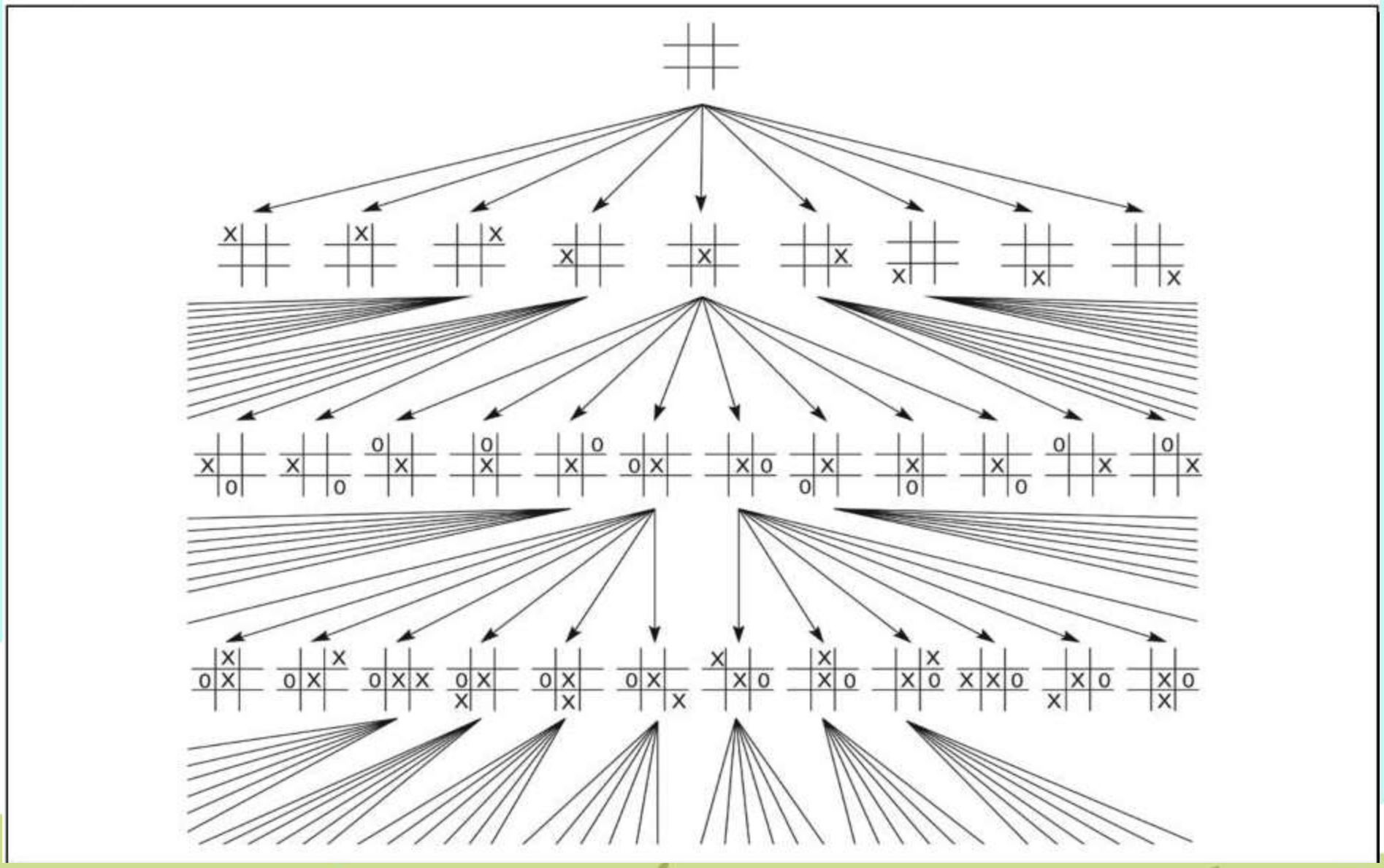
- Labeled Graph
- Labels on arcs
- Directed graph
- Path
- Rooted Graph
- Tree\Relation between nodes
  - Parent, child, siblings;
  - Ancestral & Descendant nodes
- Leaf node

State Space Representation of Problems

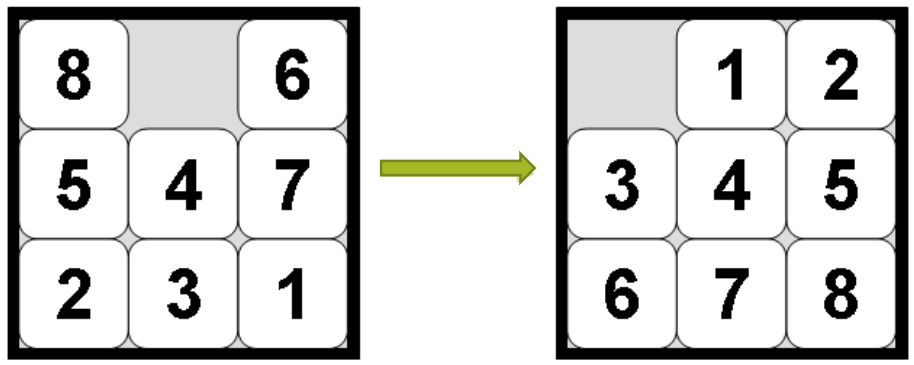
[N, A, S, GD]

- N- nodes/ states
- A- Arcs/steps
- S- Start node
- GD- Goal State
- Solution – Path from S to GD

State Space Representation of Tic-Tac-Toe

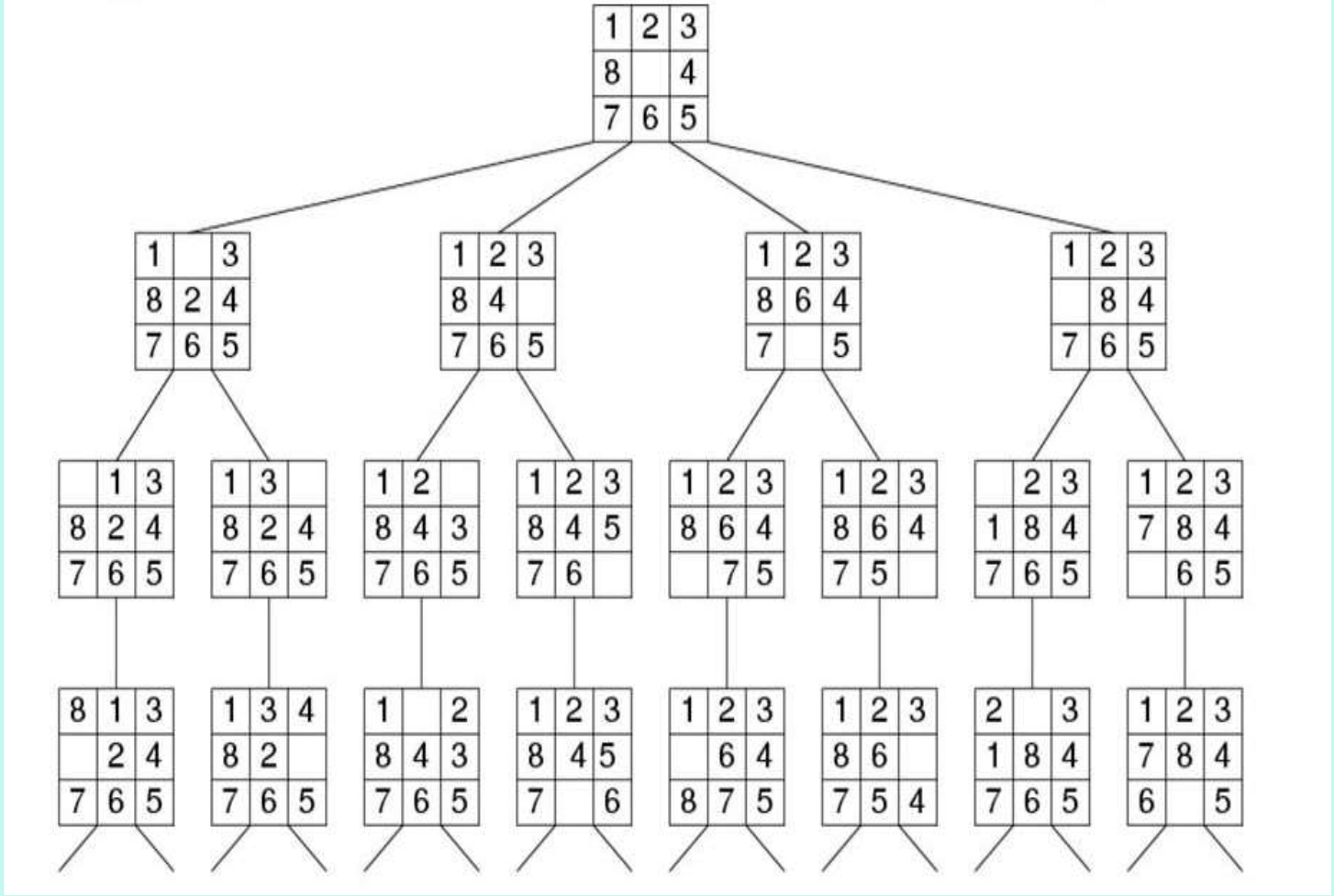


State Space Representation of 8-puzzle



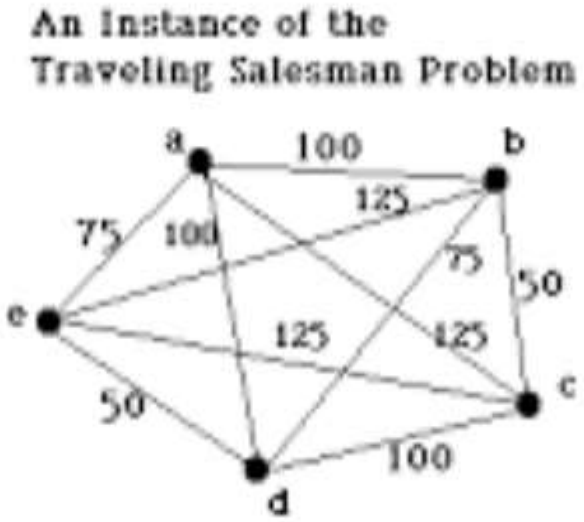
Initial State

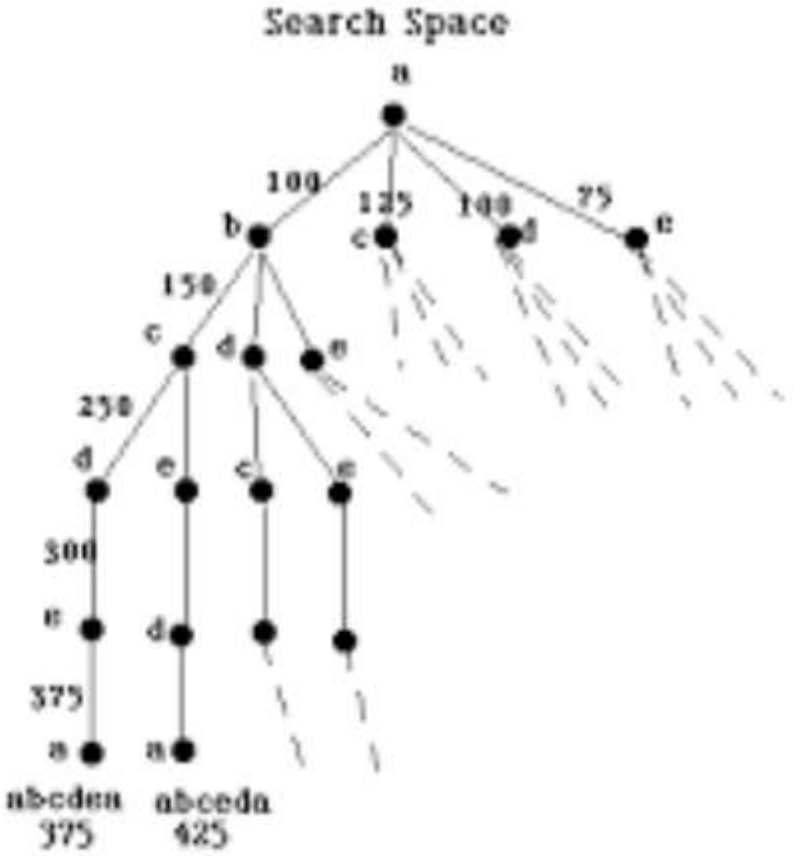
Goal State



The Traveling Salesman Problem

- Starting from city 1, the salesman must travel to all cities once before returning home
- The distance between each city is given, and is assumed to be the same in both directions
- Only the links shown are to be used
- Objective - Minimize the total distance to be travelled





Simplifying AI Models with the PEAS Representation System

PEAS => Performance, Environment, Actuators, and Sensors.

PEAS system for a ketchup-producing industry

Agent: Tomato classification system.

Sensors: Weighing sensors, Cameras for visual input, color sensing, etc.

Actuators: Track changing mechanism for segregation, display boards, or a Y-belt for quick classification into ripe and unripe tomatoes.

Environment: Our environment can be a moving walkway through which the tomatoes are passed on for segregation. It should have a good source of light for better camera input.

Performance: It measures how successful the agent is in classifying the tomatoes. It can be a confusion matrix with true positive, true negative, false positive, and false negative numbers or the model's accuracy.

Strategies for State Space Search

- Data-Driven and Goal Driven Search
- Graph Search
- Depth-First & Breadth-First Search
- Depth First with iterative Deepening

STATE SPACE SEARCH

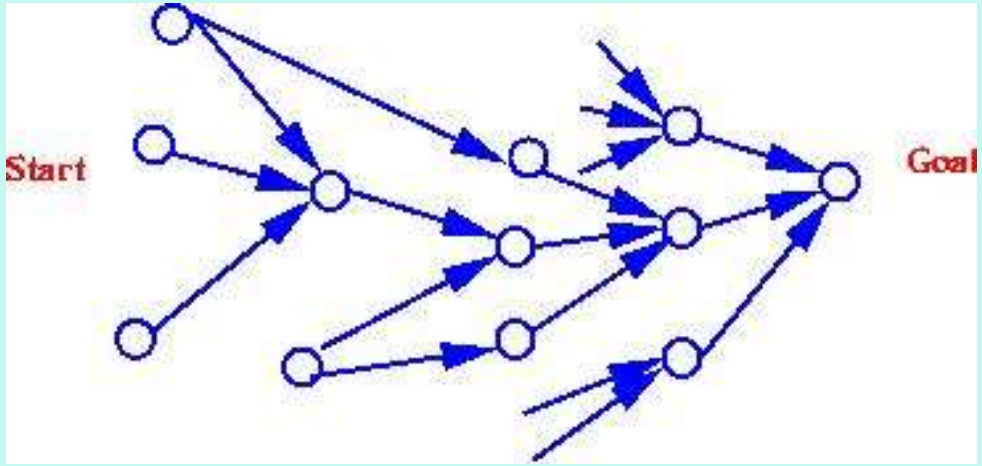
Data-Driven Search

- Data Driven approach collects the facts about the problem
- Applies rules on the data and produce new facts from it to move towards the goal.
- It is also termed as forward chaining.

Goal Driven Search

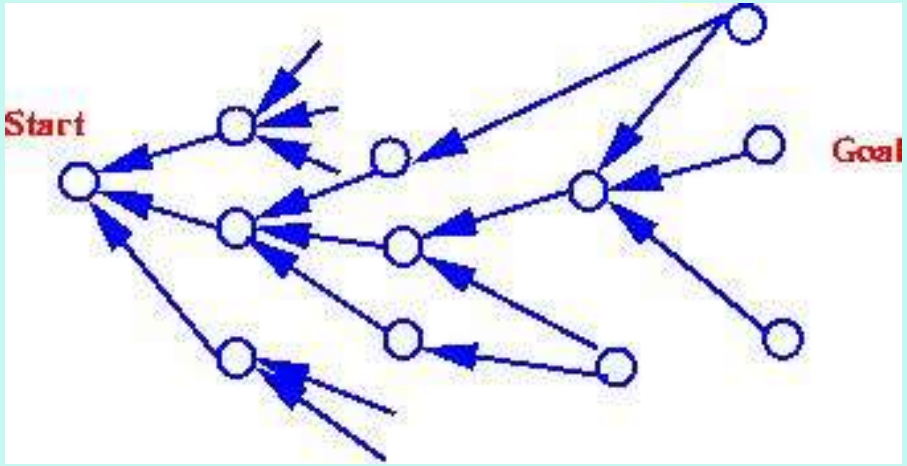
- Goal driven approach focus on the goal
- Search the facts and rules that could be able to produce that goal.
- It works by processing in backward direction via rules and sub goals to reach to the facts of the problem.
- It is also called backward chaining.

Forward Chaining.



- A goal or hypothesis is given in the problem statement
- There are a large number of rules that match the facts of the problem => an increasing number of conclusions or goals
- Problem data are not given but must be acquired by the problem solver

Backward Chaining.

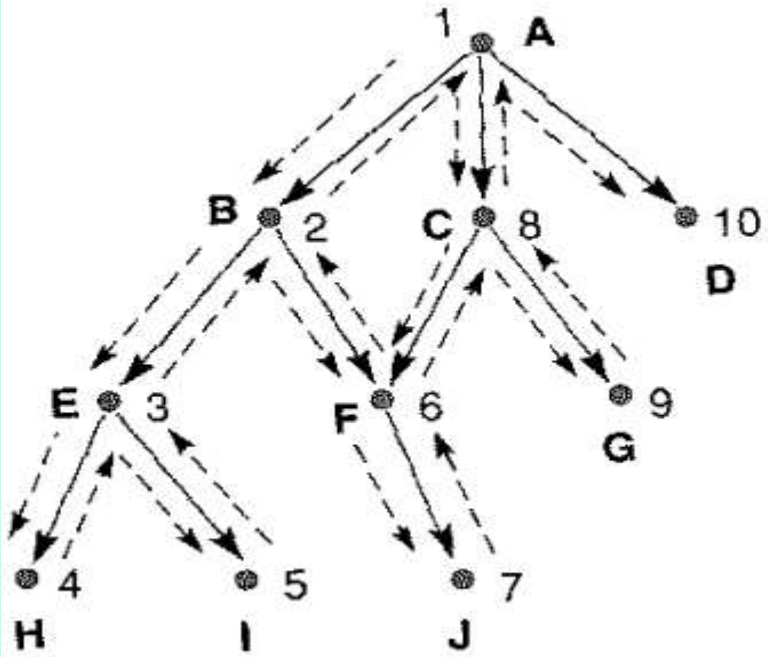


- All or most of the data are given in the initial problem statement.
- There are a large number of potential goals, but there are only a few ways to use the facts and given information of a particular problem instance.
- It is difficult to form a goal or hypothesis.

Implementing Graph Search

- a problem solver must find a path from a start state to a goal through the state space graph.
- Backtracking is a technique for systematically trying all paths through a state space
  - Begins at the start state and pursues a path until it reaches either a goal or a "dead end.“
  - Goal ➔ it quits and returns the solution path.
  - Dead end ➔ "backtracks" to the most recent node on the path having unexamined siblings and continues down one of these branches,

```
function backtrack;
begin
   SL := [Start];  NSL := [Start];  DE := [ ];  CS := Start;                    % initialize:
   while NSL π [ ] do                                  % while there are states to be tried
      begin
         if CS = goal (or meets goal description)
            then return SL;                          % on success, return list of states in path.
         if CS has no children (excluding nodes already on DE, SL, and NSL)
            then begin
               while SL is not empty and CS = the first element of SL do
                  begin
                     add CS to DE;                                % record state as dead end
                     remove first element from SL;                                %backtrack
                     remove first element from NSL;
                     CS := first element of NSL;
                  end
               add CS to SL;
            end
            else begin
               place children of CS (except nodes already on DE, SL, or NSL) on NSL;
               CS := first element of NSL;
               add CS to SL
            end
      end;
      return FAIL;
end.
```



Initialize: SL = [A]; NSL = [A]; DE = [ ]; CS = A;

AFTER

ITERATION	CS	SL	NSL	DE
0	A	[A]	[A]	[ ]
1	B	[B A]	[B C D A]	[ ]
2	E	[E B A]	[E F B C D A]	[ ]
3	H	[H E B A]	[H I E F B C D A]	[ ]
4	I	[I E B A]	[I E F B C D A]	[H]
5	F	[F B A]	[F B C D A]	[E I H]
6	J	[J F B A]	[J F B C D A]	[E I H]
7	C	[C A]	[C D A]	[B F J E I H]
8	G	[G C A]	[G C D A]	[B F J E I H]

BACKTRACKING

Usage of Backtracking in Graph Search

I. The use of a list of unprocessed states (NSL) to allow the algorithm to return (backtrack) to any of these states.

2. A list of “bad" states (DE) to prevent the algorithm from retrying useless paths.

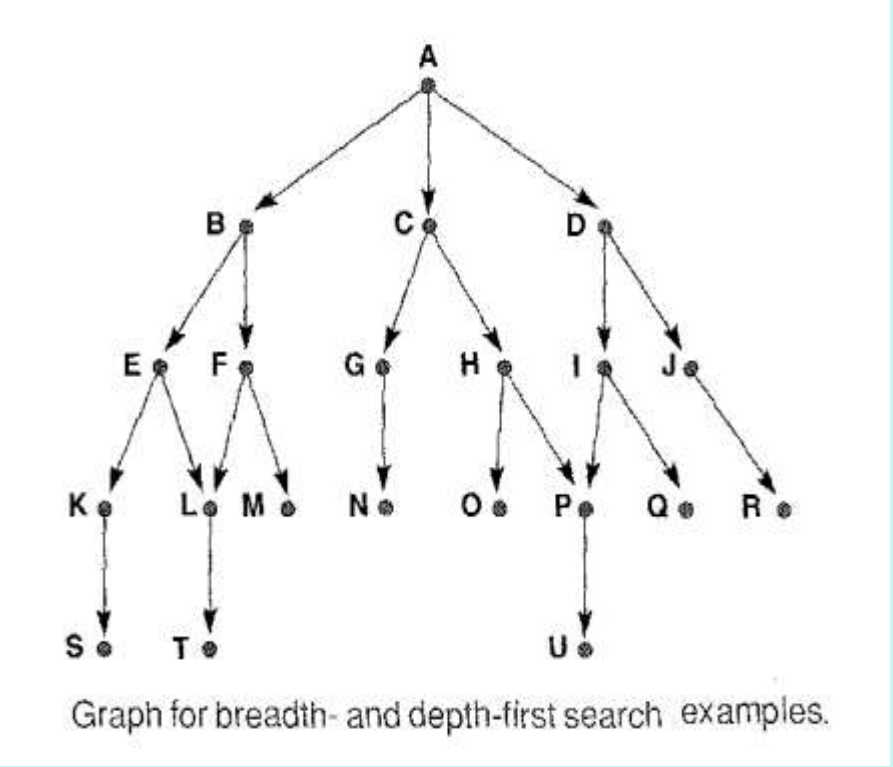
3. A list of nodes (SL) on the current solution path that is returned if a goal is found.

4. Explicit checks for membership of new states in these lists to prevent looping.

Breadth-First and Depth-First Search

BFS

OPEN	CLOSED
A	
BCD	A
CDEF	BA
DEFGH	CBA
EFGHIJ	DCBA
FGHIJKL	EDCBA



Graph for breadth- and depth-first search examples.

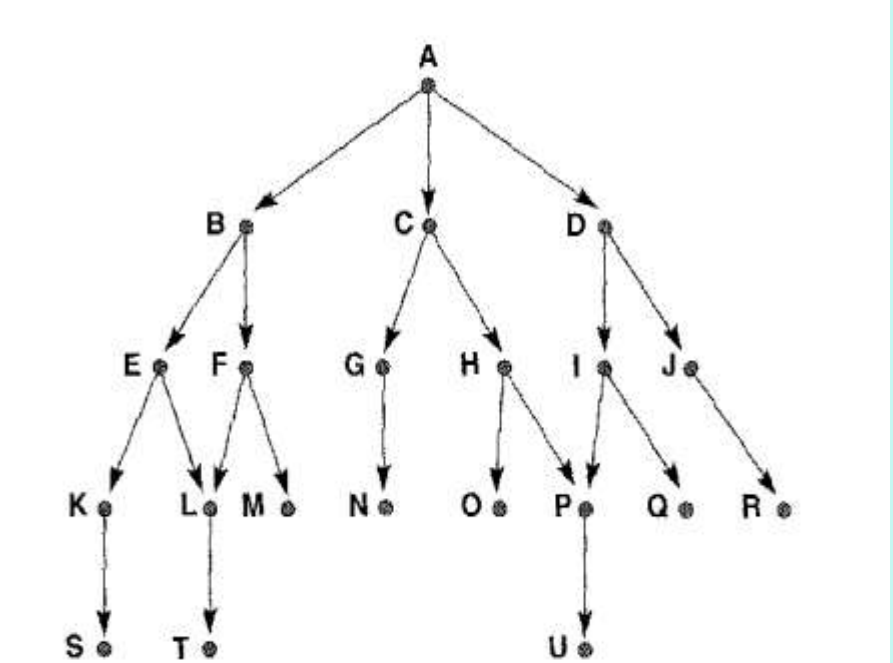
DFS

OPEN	CLOSED
A	
BCD	A
EFCD	BA
KLFCD	EBA
SLFCD	KEBA

Breadth-First and Depth-First Search

BFS

OPEN	CLOSED
A	
BCD	A
CDEF	BA
DEFGH	CBA
EFGHIJ	DCBA
FGHIJKL	EDCBA
GHIJKLM	FEDCBA
HIJKLMN	GFEDCBA
IJKLMNOP	HGFEDCBA
JKLMNOPQ	IHGFEDCBA
KLMNOPQR	JIHGFEDCBA
LMNOPQRS	KJIHGFEDCBA
MNOPQRST	LKJIHGFEDCBA
NOPQRST	MLKJIHGFEDCBA
OPQRST	NMLKJIHGFEDCBA
PQRST	ONMLKJIHGFEDCBA
QRSTU	PONMLKJIHGFEDCBA
RSTU	QPONMLKJIHGFEDCBA
STU	RQPONMLKJIHGFEDCBA
TU	SRQPONMLKJIHGFEDCBA
U	TSRQPONMLKJIHGFEDCBA



Graph for breadth- and depth-first search examples.

DFS

OPEN	CLOSED
A	
BCD	A
EFCD	BA
KLFCD	EBA
SLFCD	KEBA
LFCD	SKEBA
TFCD	LSKEBA
FCD	TLSKEBA
MCD	FTLSKEBA
CD	MFTLSKEBA
GHD	CMFTLSKEBA
NHD	GCMFTLSKEBA
HD	NGCMFTLSKEBA
OPD	HNGCMFTLSKEBA
PD	OHNGCMFTLSKEBA
UD	POHNGCMFTLSKEBA
D	UPOHNGCMFTLSKEBA
IJ	DUPOHNGCMFTLSKEBA
QJ	IDUPOHNGCMFTLSKEBA
J	QIDUPOHNGCMFTLSKEBA
R	JQIDUPOHNGCMFTLSKEB

Breadth-First and Depth-First Search

```
function breadth_first_search;

begin
   open := [Start];
   closed := [ ];
   while open π [ ] 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.
```

```
function depth_first_search;

begin
   open := [Start];
   closed := [ ];
   while open π [ ] 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;
   return FAIL
end.
```

Breadth-first Search (BFS)	Depth-first Search (DFS)
Uses queue data structure to store the nodes to be visited	Uses stack to store the nodes to be visited
Traverses a graph level-wise	Traverses a graph depth-wise
Not suitable for decision-making trees used in games or puzzle	More suitable for game or puzzle problems
More suitable when the target node is closer to the source	More suitable when the target node is away from the source node

DFS or Depth First Search

- On unweighted graph, DFS will create minimum spanning tree for all pair shortest path tree
- We can detect cycles in a graph using DFS.
- Using DFS we can find path between two given vertices u and v.
- We can perform topological sorting is used to scheduling jobs from given dependencies among jobs.
- Using DFS, we can find strongly connected components of a graph. If there is a path from each vertex to every other vertex, that is strongly connected.

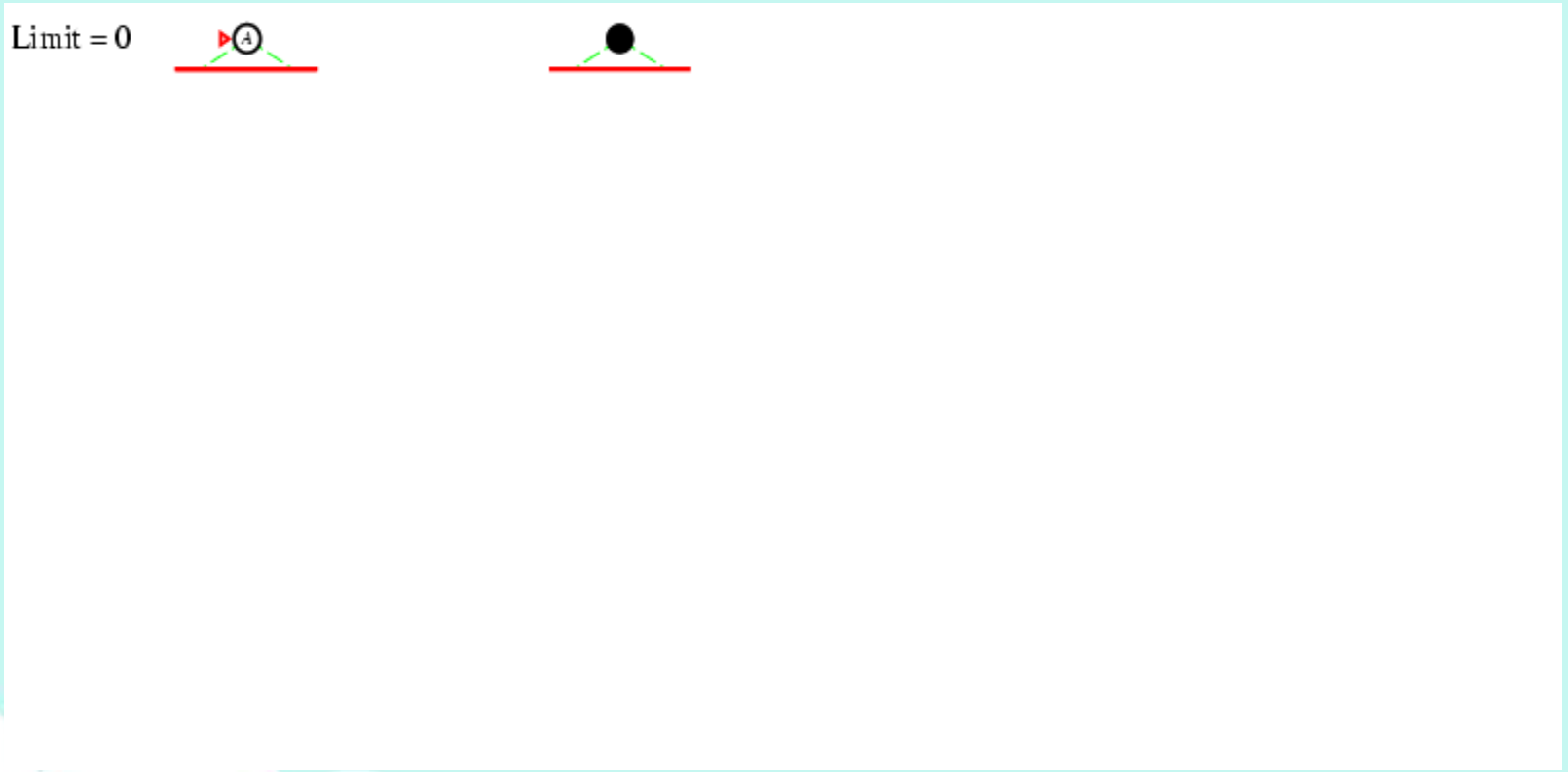
BFS (Breadth First Search)

- In peer-to-peer network like bit-torrent, BFS is used to find all neighbor nodes
- Search engine crawlers are used BFS to build index. Starting from source page, it finds all links in it to get new pages
- Using GPS navigation system BFS is used to find neighboring places.
- In networking, when we want to broadcast some packets, we use the BFS algorithm.
- Path finding algorithm is based on BFS or DFS.
- BFS is used in Ford-Fulkerson algorithm to find maximum flow in a network.

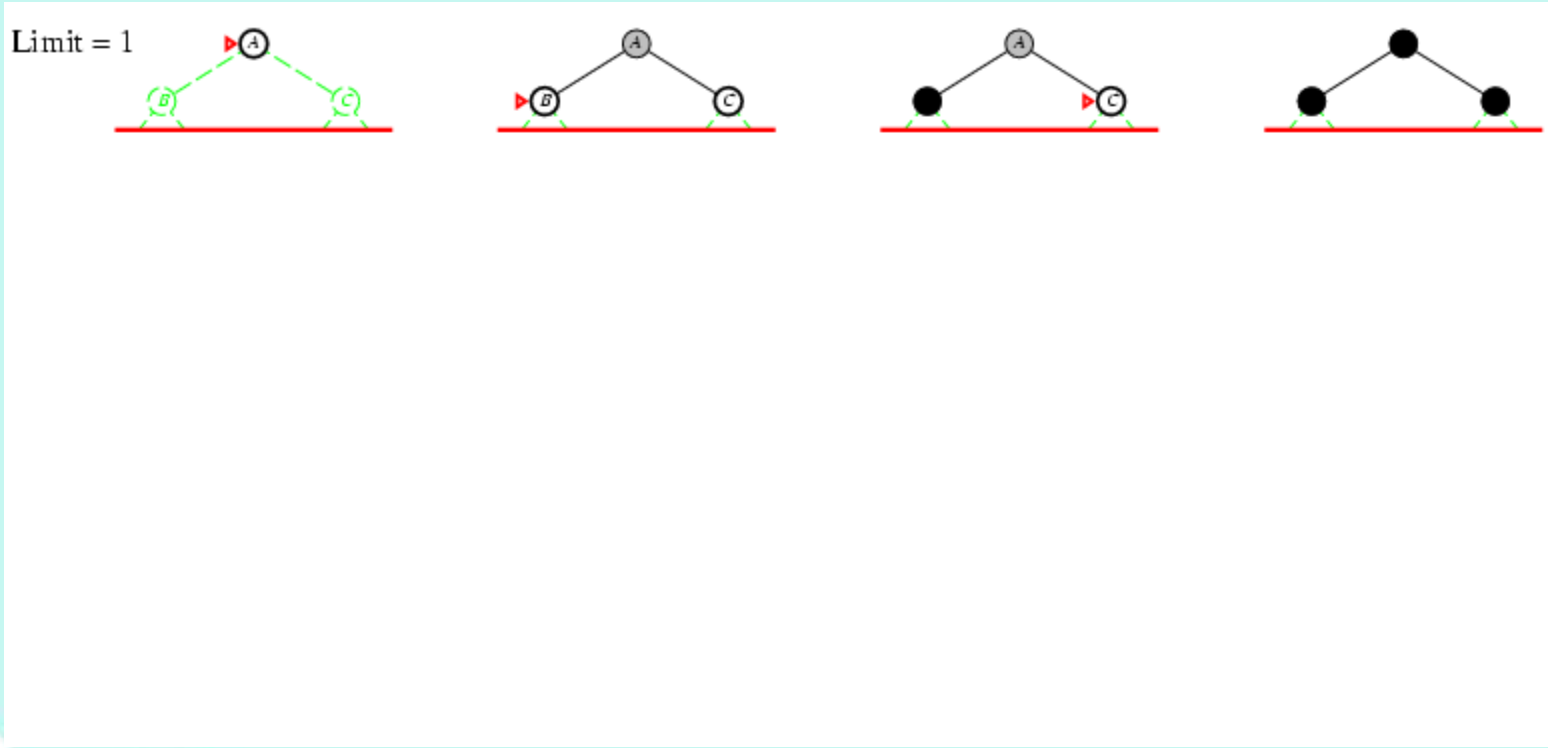
Iterative deepening search

- Use DFS as a subroutine
  1. Check the root
  2. Do a DFS searching for a path of length 1
  3. If there is no path of length 1, do a DFS searching for a path of length 2
  4. If there is no path of length 2, do a DFS searching for a path of length 3…

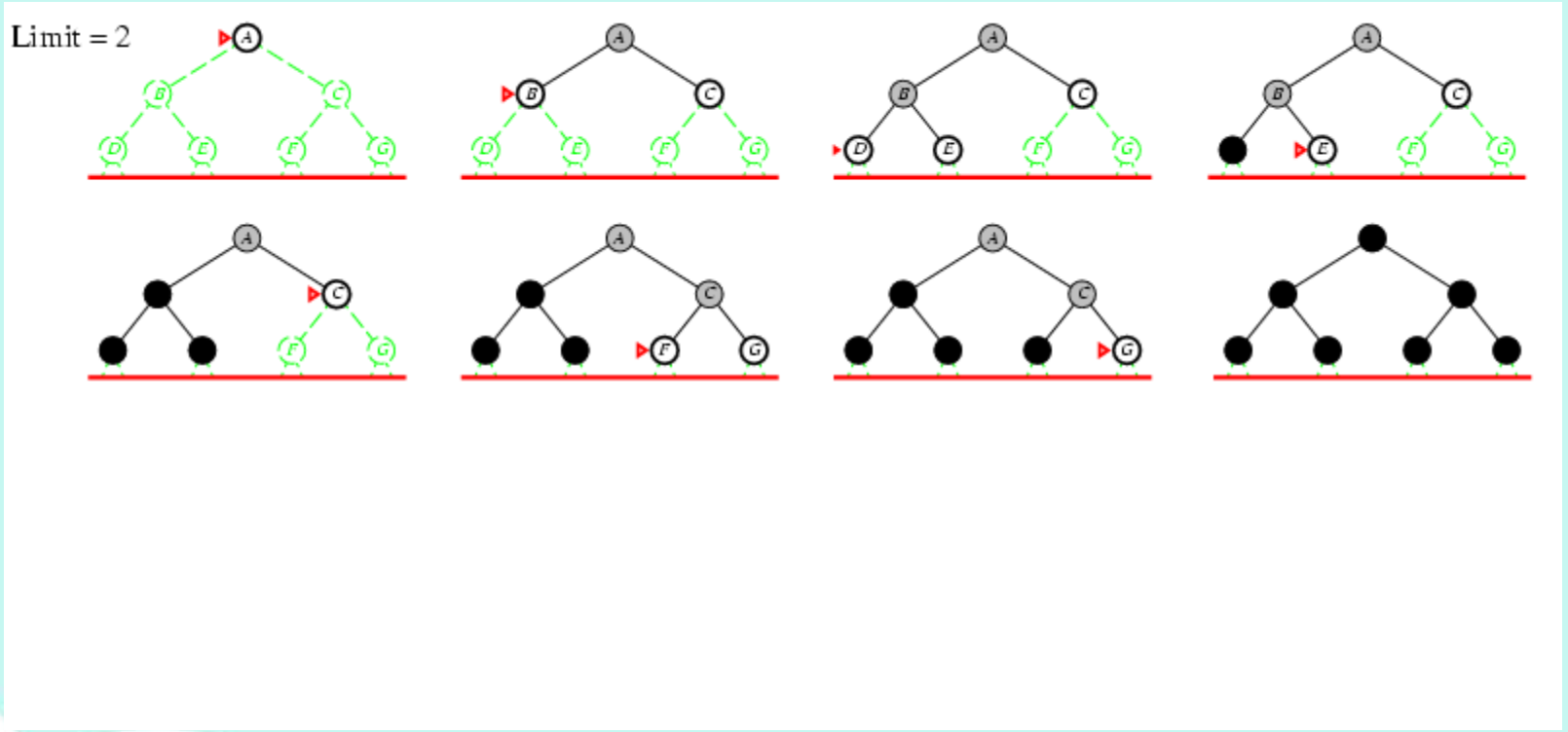
Iterative deepening search



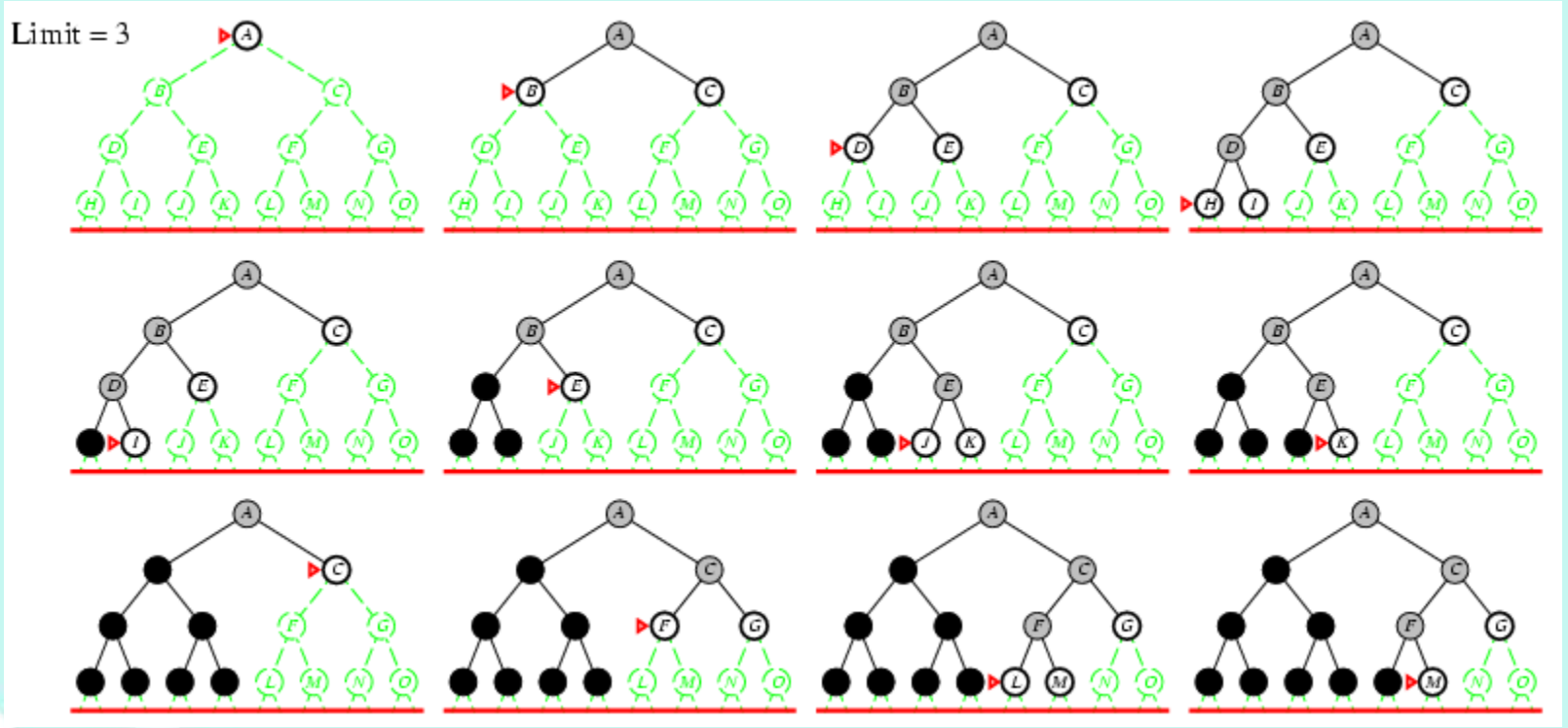
Iterative deepening search

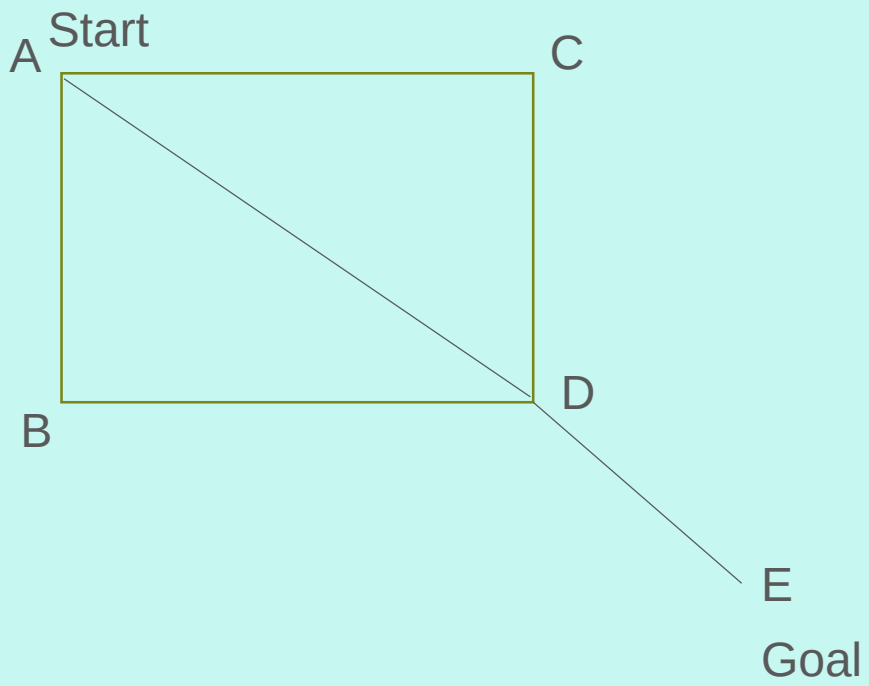


Iterative deepening search



Iterative deepening search




A
Start
C
B
D
E
Goal

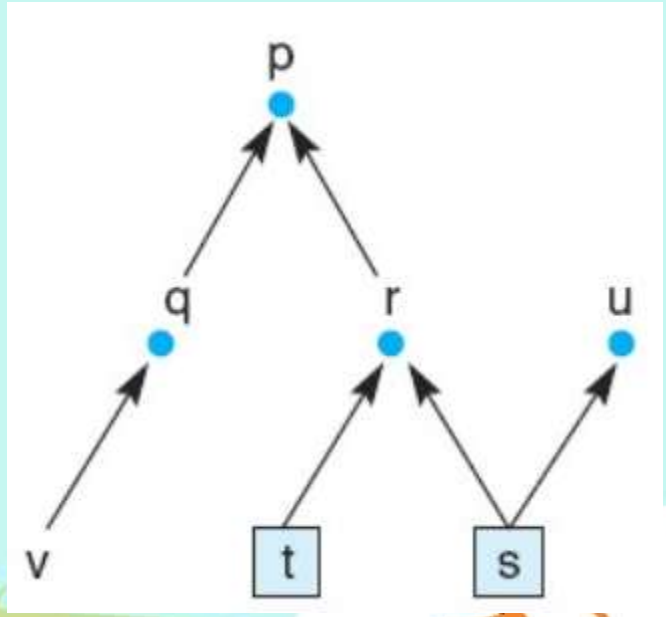
Using the State Space to Represent Reasoning with the Predicate Calculus

- Representation
  - Logical expressions as **states**
  - Inference Rules as **links**
- Correctness
  - **Soundness** and **Completeness** of Predicate Calculus **inference Rules** guarantee the **Correctness** of conclusions
- Theorem Proof
  - State space Search

Example

- q → p
- r → p
- s → r
- t → r
- s → u
- v →q
- s
- t

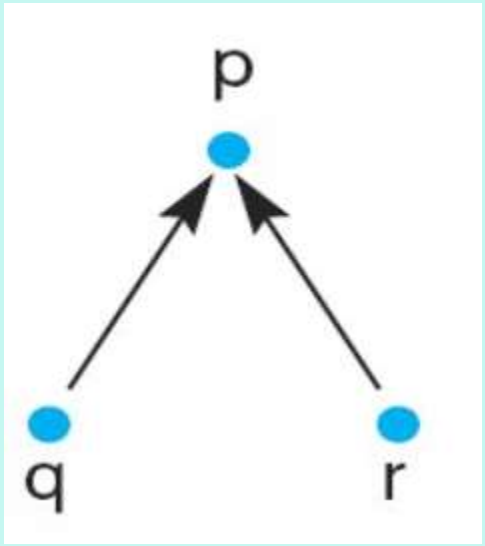
- Nodes ➔ letters
- Links ➔ Implications

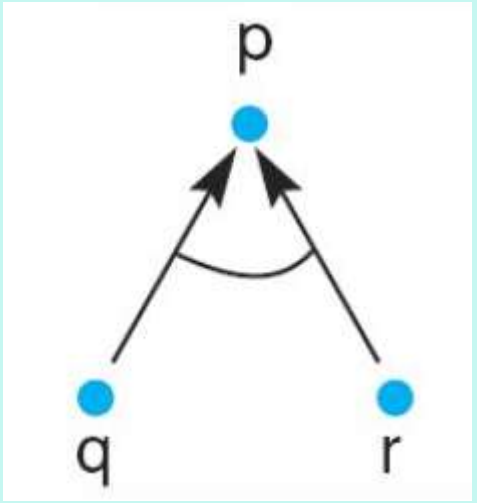


Soundness is the property of only being able to prove "true" things. Completeness is the property of being able to prove all true things.

And/Or Graph

- OR – Separate
- AND – connected





q v r → p

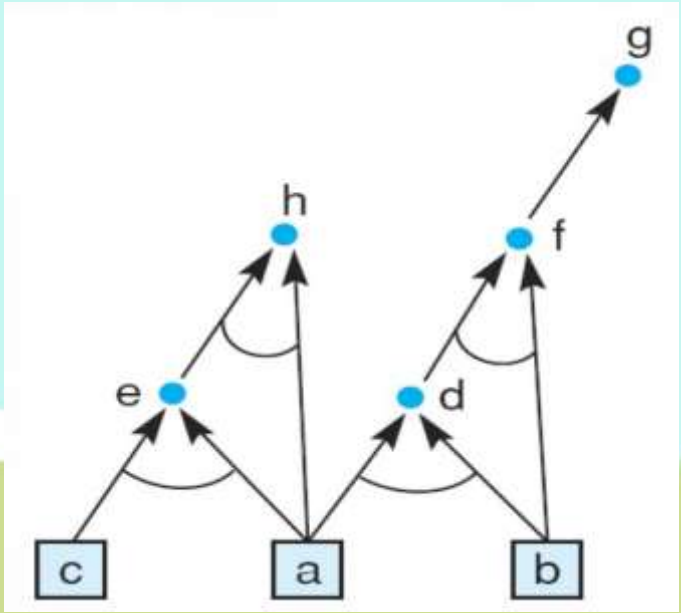
q ∧ r → p

HYPERGRAPH

- A hypergraph consists of :
  - N, a set of nodes
  - H, a set of hyperarcs 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
- Ordinary graph – Special case of hypergraph in which all sets of descendant nodes have a cardinality of 1
- k-connectors – k is the cardinality of the set of descendant nodes

Example:

- a
- b
- c
- a Λ b→ d
- a Λ c→ e
- b Λ d → f
- F → g
- a Λ e → h



Example 1:

1. Fred is a collie.
2. Sam is Fred’s master.
3. The day is Saturday.
4. It is cold on Saturday.
5. Fred is trained.
6. Spaniels are good dogs and so are trained collies.
7. If a dog is a good dog and has a master then he will be with his master.
8. If it is Saturday and warm, then Sam is at the park.
9. If it is Saturday and not warm, then Sam is at the museum..

Draw the AND/OR graph using the data-driven strategy to find the location of fred, i.e., evaluate the goal location(fred,X).

Example 1:

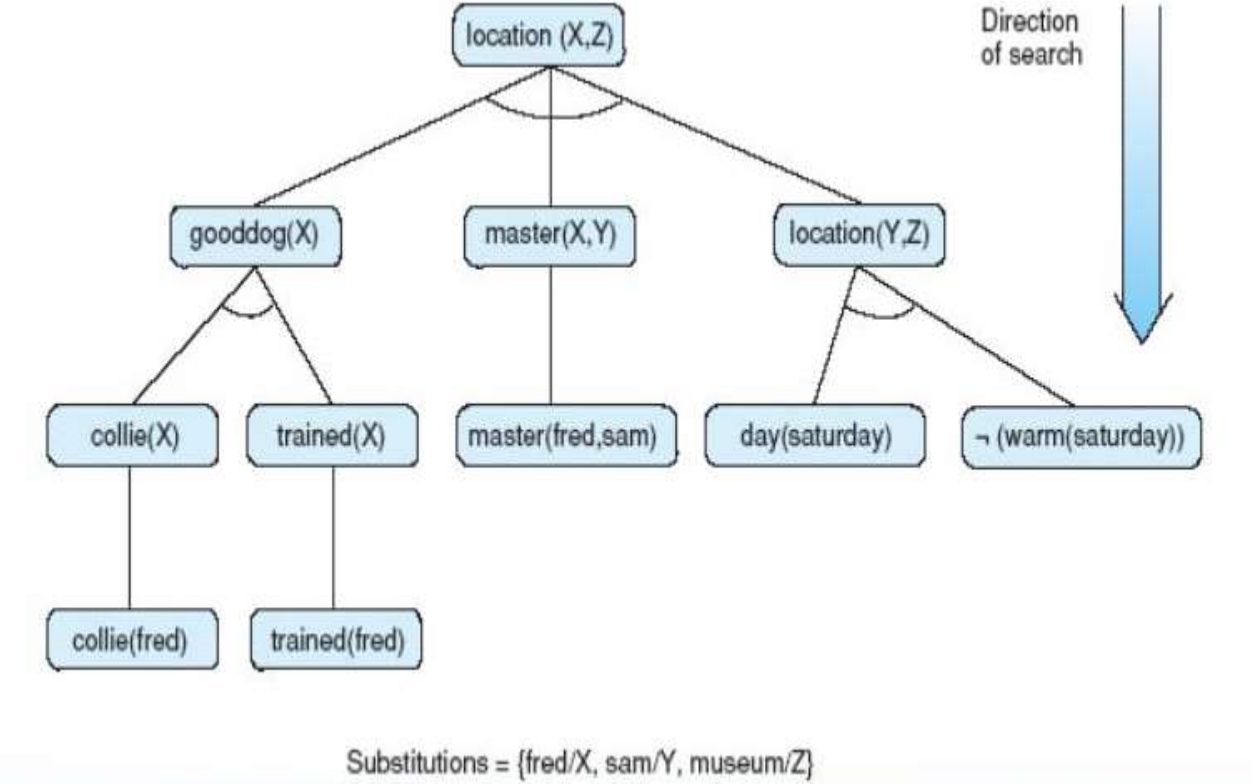
1. Fred is a collie.
2. Sam is Fred's master.
3. The day is Saturday.
4. It is cold on Saturday.
5. Fred is trained.
6. Spaniels are good dogs and so are trained collies.
7. If a dog is a good dog and has a master then he will be with his master.
8. If it is Saturday and warm, then Sam is at the park.
9. If it is Saturday and not warm, then Sam is at the museum..

1. **collie(fred).**
2. **master(fred,sam).**
3. **day(saturday).**
4. **¬ (warm(saturday)).**
5. **trained(fred).**
6. **∀ X[spaniel(X) ∨ (collie(X) ∧ trained(X)) → gooddog(X)]**
7. **∀ (X,Y,Z) [gooddog(X) ∧ master(X,Y) ∧ location(Y,Z) → location(X,Z)]**
8. **(day(saturday) ∧ warm(saturday)) → location(sam,park).**
9. **(day(saturday) ∧ ¬ (warm(saturday))) → location(sam,museum).**

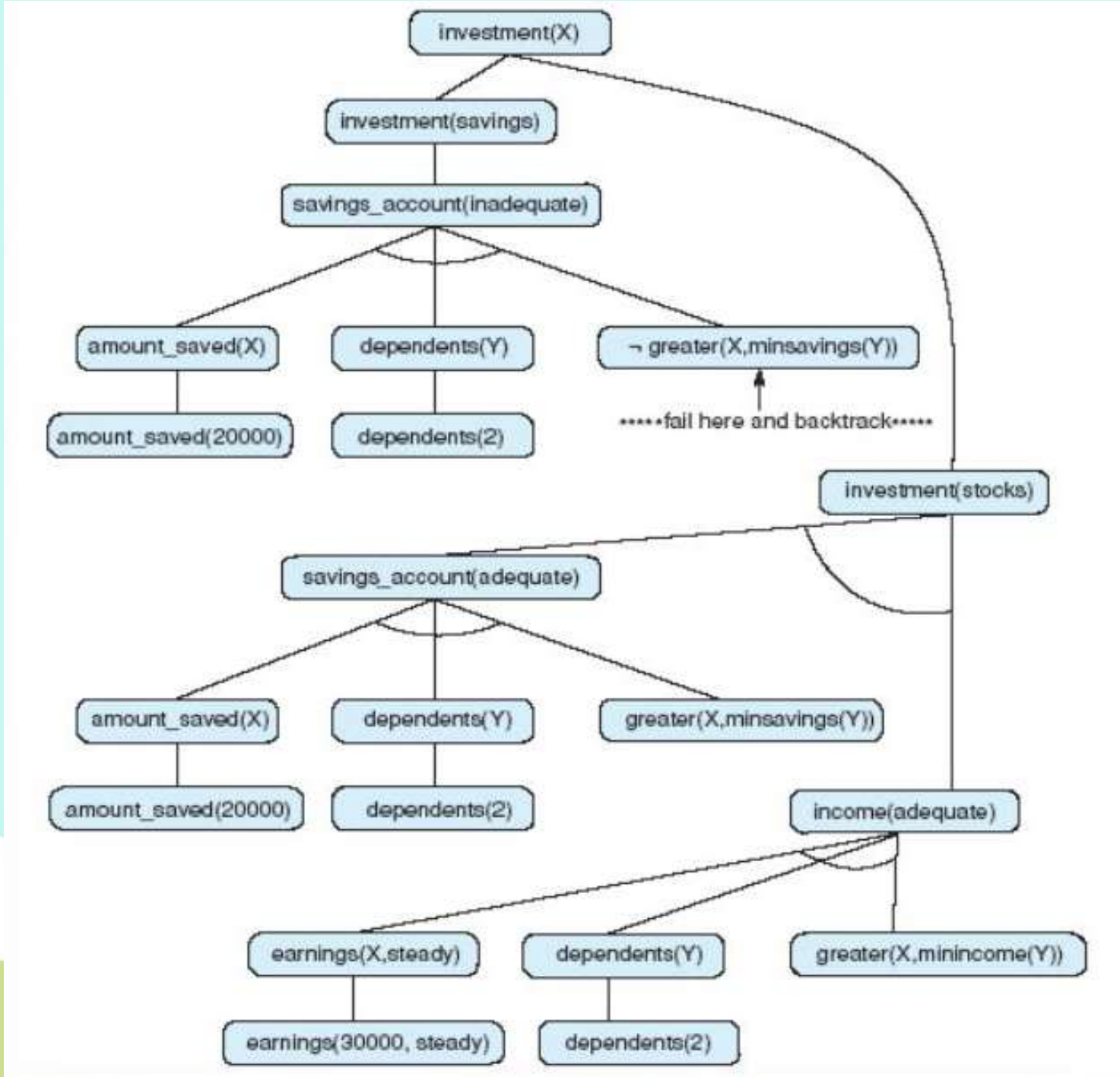
Draw the AND/OR graph using the data-driven strategy to find the location of fred, i.e., evaluate the goal location(fred,X).

AND/OR Graph for Example 1

1. collie(fred).
2. master(fred,sam).
3. day(saturday).
4. ¬ (warm(saturday)).
5. trained(fred).
6. ∀ X[spaniel(X) ∨ (collie(X) ∧ trained(X)) → gooddog(X)]
7. ∀ (X,Y,Z) [gooddog(X) ∧ master(X,Y) ∧ location(Y,Z) → location(X,Z)]
8. (day(saturday) ∧ warm(saturday)) → location(sam,park).
9. (day(saturday) ∧ ¬ (warm(saturday))) → location(sam,museum).

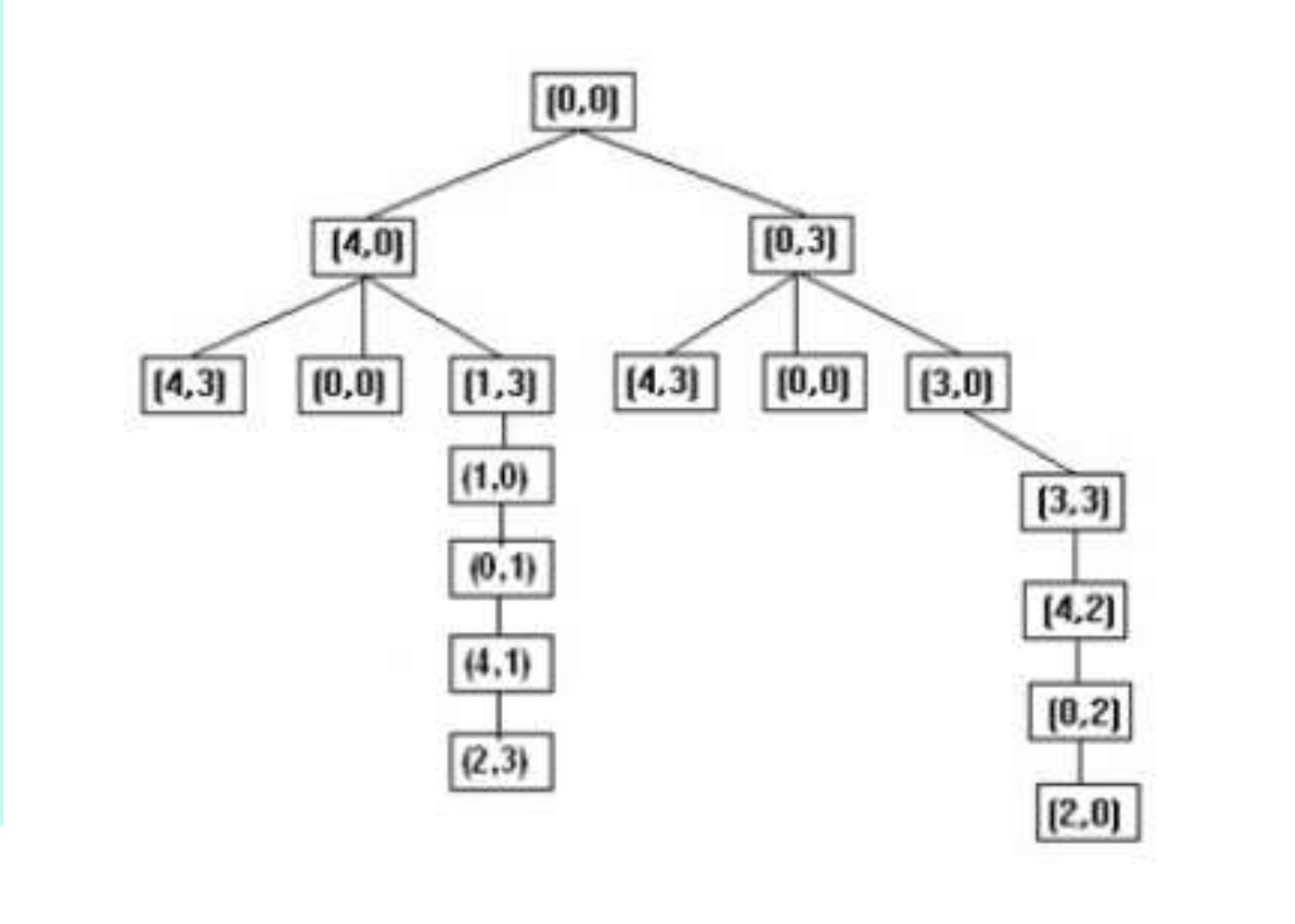


AND/OR GRAPH FOR FINANCIAL ADVISOR PROBLEM Example 2



WATER JUG PROBLEM

- You are given two jugs, a 4-gallon one and a 3-gallon one. Neither has any measuring mark 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 the 4-gallon jug?


(0,0)
(4,0)
(0,3)
(4,3)
(0,0)
(1,3)
(4,3)
(0,0)
(3,0)
(1,0)
(3,3)
(0,1)
(4,2)
(4,1)
(0,2)
(2,3)
(2,0)