Artificial intelligence

What is Artificial intelligence?

- It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence.

- “Intelligence implies that a machine must be able to adapt to new situations”

- Ability to learn
- Ability to think abstractly
- To solve problems
- To percieve relationship
- To adjust to one’s environment
- To profit by experience

- Woodworth → intelligence is a way of acting.
- Woadrow → intelligence is an acquiring capacity
- Binet → comprehension, invention , direction and criticism– intelligence contained in these four words.
- Ryburn → intelligence is the power which enables us to solve problems and to achieve our purpose.

- Intelligence is not a single power or capacity or abilitiy which operates equally well in all situations.
- It is rather than composite of several different abilities.

What is the objective of “AI”

One term is

- “the ability to reason, to trigger new thoughts, to perceive and learn is intelligence”.

Second term is

“thought”

A thought is a mechanism which

1. Stimulates
   a. action
   b. further thought
   c.information generation
   d. knowledge generation

2. Is triggered by

a. External stimulus or

b. internal stimulus

3. Acts through

a. Present environment

b. past memory

4. Is stored as

a. charged /discharged state of neurons.

b. electromagnetic thought waves

Definition of AI

- “John McCarthy “ gives in 1956 “Developing computer programs to solve complex problems by applications of processes that are analogous to human reasoning processes
- “Ai is the branch of computer science that is concerned with the automation of intelligent behavior.”
- AI is the study of how to make computers do things which, at the moment, people do better.

- the intelligent is behavior , when we call this man Intelligent, we mean by that (he have the ability to Think, understand, learn and make decision) so if we a combine this word with system to become (Intelligent System(IS))we mean by that , the system able to (Think, understand, learn and make decision) in other word :

Definitions of AI:
systems that

- think like humans
- act like humans
- think rationally
- act rationally

AI Tree

Fruits: Applications

Branches: Expert Systems, Natural Language processing, Speech Understanding, Robotics and Sensory Systems, Computer Vision, Neural Computing, Fuzzy Logic, GA

Roots: Psychology, Philosophy, Electrical Engg, Management Science, Computer science, Linguistics

- **Artificial Intelligence**
  *primary symbolic process
  *Heuristic search
  -steps are implicit (hidden)
  *usually easy to modify update and enlarge
  *some incorrect answers are tolerable
  *satisfactory answers usually acceptable
- **Conventional Programming**
  *numeric
  *Algorithmic--steps are explicit (open)
  *information and control are integrated together
  *difficult to modify
  *correct answers are required
  *best possible solution usually sought (required)

Difference between AI & conventional S/W

Features	AI programs	Conventional s/w
Processing type	Symbolic type	Numeric
Technique used	Heuristic search	Algorithm search
Solutions steps	Indefinite	definite
Answers sought	Satisfactory	Optimal
Knowledge	Imprecise	Precise
Modification	Frequent	Rare
Involves	Large knowledge	Large DB
Process	Inferential	repetitive

Examples of artificially intelligent systems include computer programs that perform

- medical diagnoses,
- mineral prospecting,
- legal reasoning,
- speech understanding,
- vision interpretation,
- natural-language processing,
- problem solving, and learning.
- Most of these systems are far from being perfected. Most have proved valuable, however, either as research vehicles or in specific, practical applications.

Applications of AI

- Game playing→
- Speech recognition→
- Understanding natural language→
- Computer vision→
- Expert system
- Heuristic classification

Areas of Artificial Intelligence

- . Perception
  - Machine vision
  - Speech understanding
  - Touch (*tactile* or *haptic*) sensation
- Robotics
- Natural Language Processing
  - Natural Language Understanding
  - Speech Understanding
  - Language Generation
  - Machine Translation
- Planning
- Expert Systems
- Machine Learning
- Theorem Proving
- Symbolic Mathematics
- Game Playing

How problems can be represented in AI

- Before a solution can be found the prime condition is that the problem must be very precisely defined.
- So to build a system to solve a particular problem, we need to do four things.

How problems can be represented in AI

1. Define the problem precisely. like what is initial situation, what will be the final, acceptable solutions.
2. Analyze the problem. various possible techniques for solving the problem.
3. Isolate and represent the task knowledge that is necessary to solve the problem.
4. Choose the best problem solving technique and apply it

The most common methods of problem representation in AI

State space representation

"A set of all possible states for a given problem is known as the state space of the problem."

or

"A *state space* represents a problem in terms of *states* and *operators* that change states."

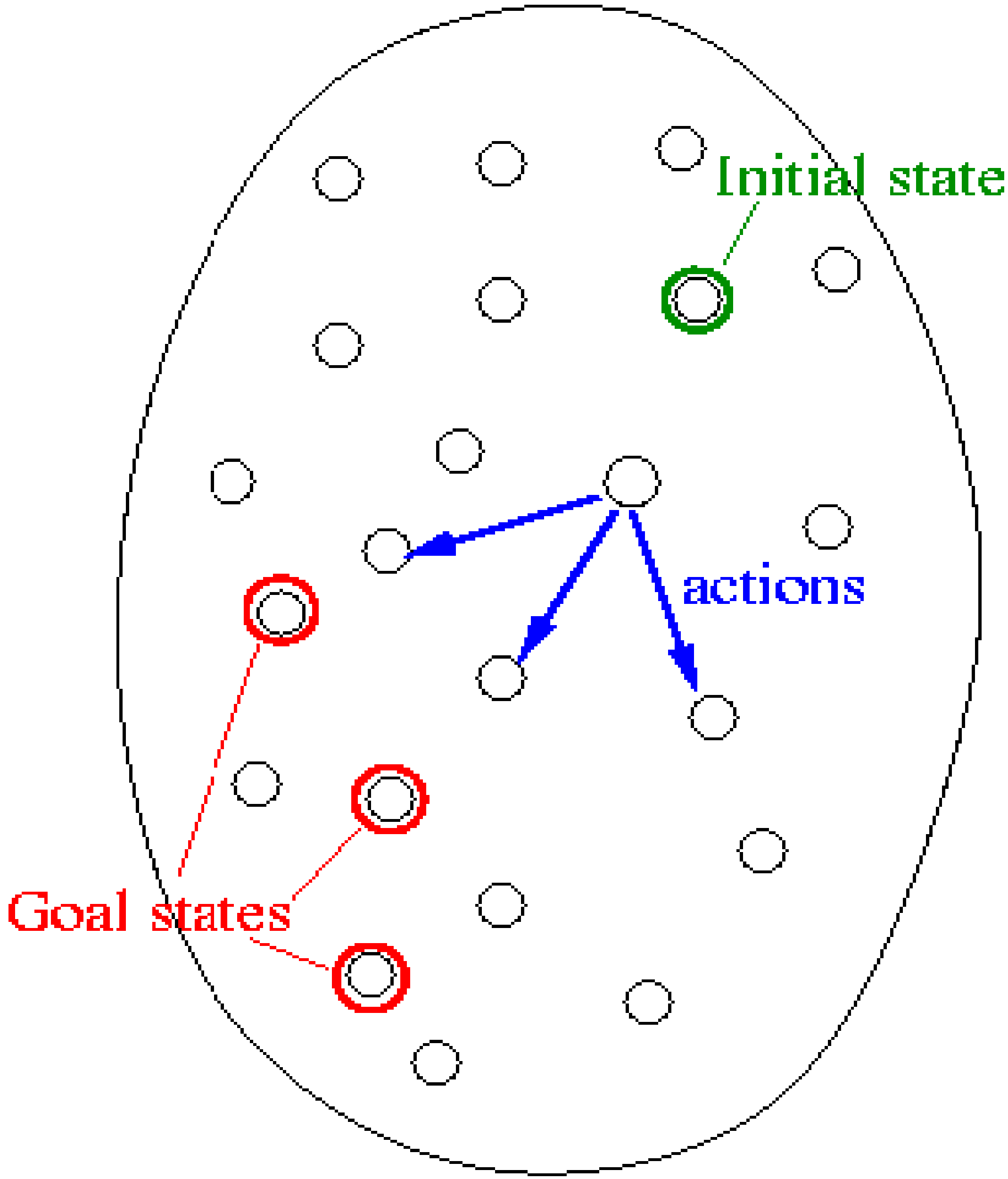
A problem space consists of

1. Precondition/An *initial state*→

2. Post condition/Final states→

3. Actions→

4. Total Cost→


Initial state
actions
Goal states

State Space Search: Summary

1. Define a state space that contains all the possible configurations of the relevant objects.

2. Specify the initial states.

3. Specify the goal states.

4. Specify a set of rules:
   - What are unstated assumptions?
   - How general should the rules be?
   - How much knowledge for solutions should be in the rules?

For example

If one wants to make a cup of coffee.

What one have to do:

→analyze the problem

→check necessary ingredients are available or not.

→if they are available.

(ii) Take some of the boiled water in a cup and add necessary amount of instant coffee powder to make decoction.

(iii) Add milk powder to the remaining boiling water to make milk.

(iv) Mix decoction and milk.

(v) Add sufficient quantity of sugar to your taste and the coffee is ready.

Now, by representing all the steps done sequentially we can make state space representation of this problem as shown in fig. 1.4.

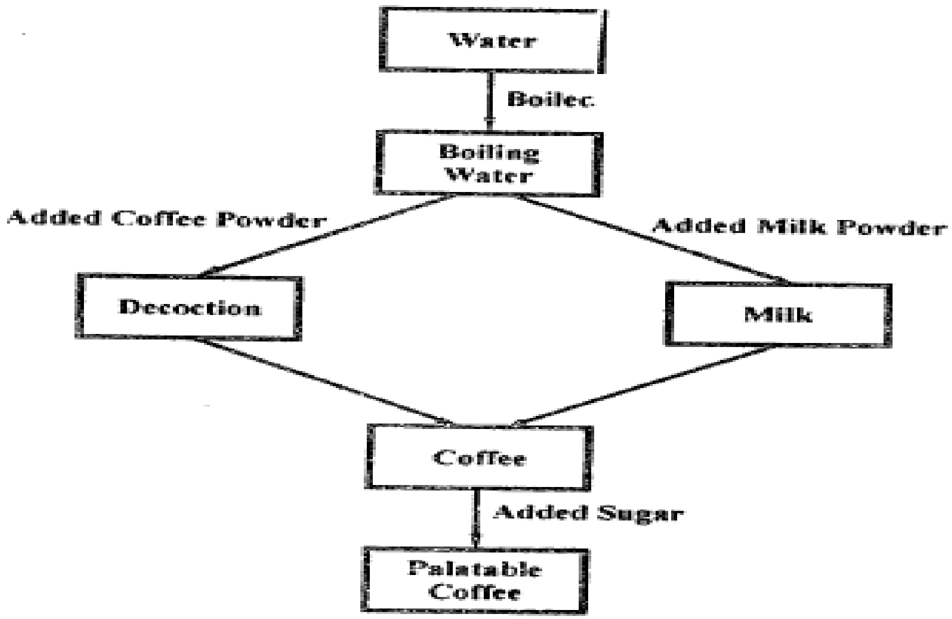


Fig. 1.4 State Space Representation of Coffee Making

Let's think about what we have done. We started with the ingredients (initial state), followed by a sequence of steps (called steps) and at last had a cup of coffee (goal state).

We added only needed amount of coffee powder, milk powder and sugar (operators).

Water jug problem?

- States– amount of water in both jugs.
- Actions—Empty large/small, pour from large/small
- Goal—specified amount of water in both jug
- Path cost—total no of actions applied

State Space Search: Playing Chess

- State space is a set of legal positions.
- Starting at the initial state.
- Using the set of rules to move from one state to another.
- Attempting to end up in a goal state.

State Space Search: Water Jug Problem

"You are given two jugs, a 4-litre one and a 3-litre 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 litres of water into 4-litre jug."

State Space Search: Water Jug Problem

- State: (x, y)

  x = 0, 1, 2, 3, or 4 y = 0, 1, 2, 3

- Start state: (0, 0).
- Goal state: (2, n) for any n.
- Attempting to end up in a goal state.

State Space Search: Water Jug Problem

1. $(x, y) \rightarrow (4, y)$
   if $x < 4$
2. $(x, y) \rightarrow (x, 3)$
   if $y < 3$
3. $(x, y) \rightarrow (x - d, y)$
   if $x > 0$
4. $(x, y) \rightarrow (x, y - d)$
   if $y > 0$

State Space Search: Water Jug Problem

5. $(x, y) \rightarrow (0, y)$

   if $x > 0$

6. $(x, y) \rightarrow (x, 0)$

   if $y > 0$

7. $(x, y) \rightarrow (4, y - (4 - x))$

   if $x + y \geq 4, y > 0$

8. $(x, y) \rightarrow (x - (3 - y), 3)$

   if $x + y \geq 3, x > 0$

State Space Search: Water Jug Problem

9. $(x, y) \rightarrow (x + y, 0)$

   if $x + y \leq 4, y > 0$

10. $(x, y) \rightarrow (0, x + y)$

    if $x + y \leq 3, x > 0$

11. $(0, 2) \rightarrow (2, 0)$

12. $(2, y) \rightarrow (0, y)$

State Space Search: Water Jug Problem

1. current state = (0, 0)

2. Loop until reaching the goal state (2, 0)
   - Apply a rule whose left side matches the current state
   - Set the new current state to be the resulting state

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

State Space Search: Water Jug Problem

The role of the condition in the left side of a rule

$\Rightarrow$ restrict the application of the rule

$\Rightarrow$ more efficient

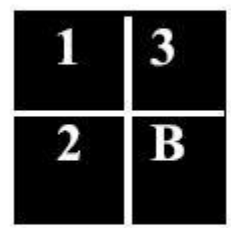
1. $(x, y) \rightarrow (4, y)$
   if $x < 4$

2. $(x, y) \rightarrow (x, 3)$
   if $y < 3$

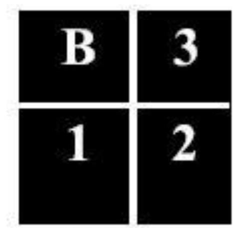
Find a driving route from city A to city B

- States– location specified by city .
- Actions– driving along the roads between cities
- Goal— city B
- Path cost—total distance or expected travel time.

- **Example**: Consider a 4-puzzle problem, where in a 4-cell board there are 3 cells filled with digits and 1 blank cell. The initial state of the game represents a particular orientation of the digits in the cells and the final state to be achieved is another orientation supplied to the game player. The problem of the game is to reach from the given initial state to the goal (final) state, if possible, with a minimum of moves. Let the initial and the final state be as shown in figures 1(a) and (b) respectively.

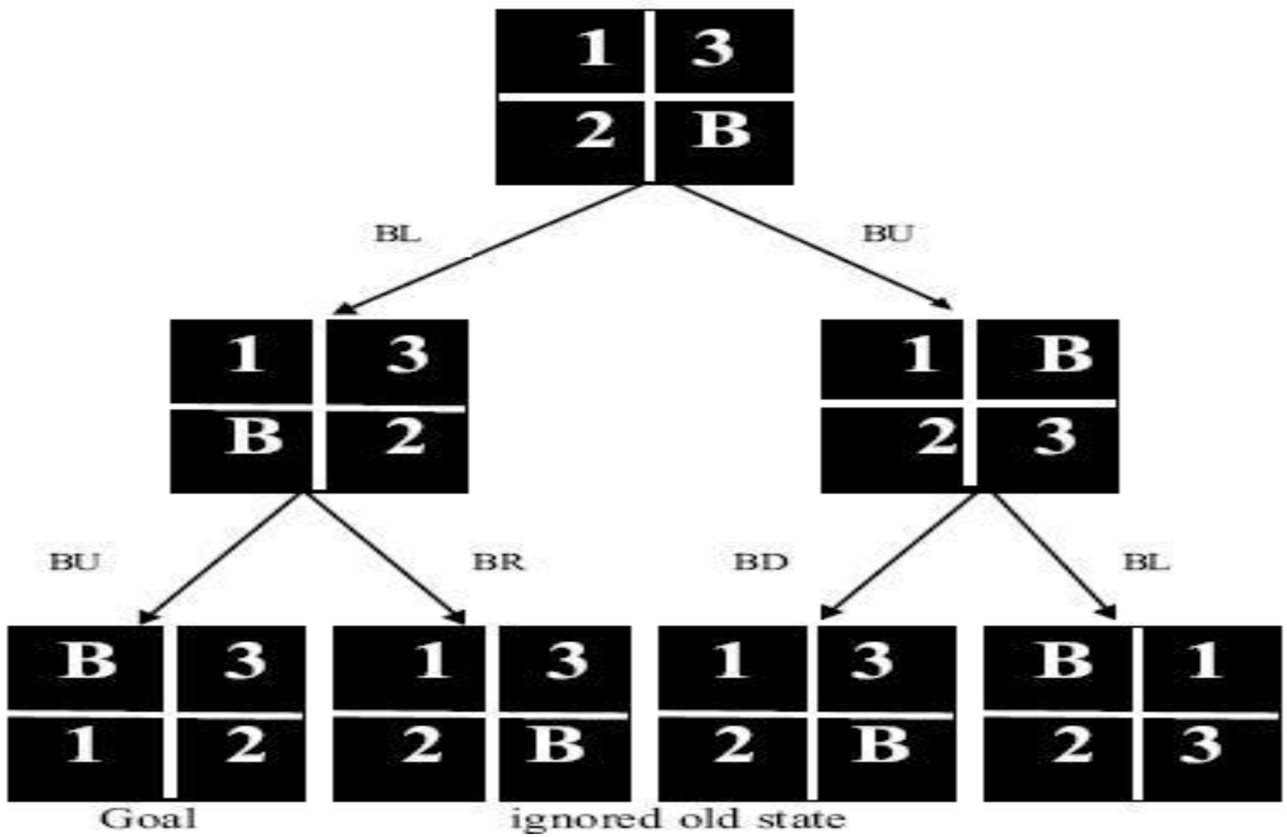


(a) initial state



(b) final state

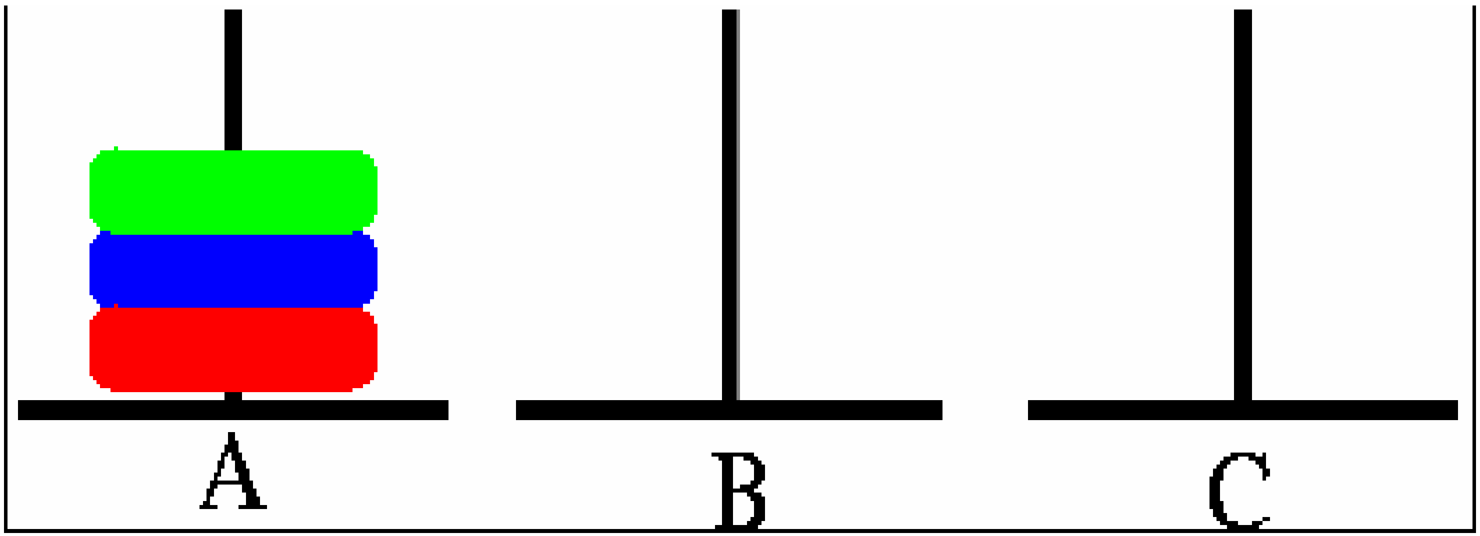
- We now define two operations, blank-up (BU) / blank-down (BD) and blank-left (BL) / blank-right (BR), and the state-space (tree) for the problem is presented below using these operators. The algorithm for the above kind of problems is straightforward. It consists of three steps, described by steps 1, 2(a) and 2(b) below.


1 3
2 B
BL
BU
1 3
B 2
1 B
2 3
BU
BR
BD
BL
B 3
1 2
1 3
2 B
1 3
2 B
B 1
2 3
Goal
ignored old state

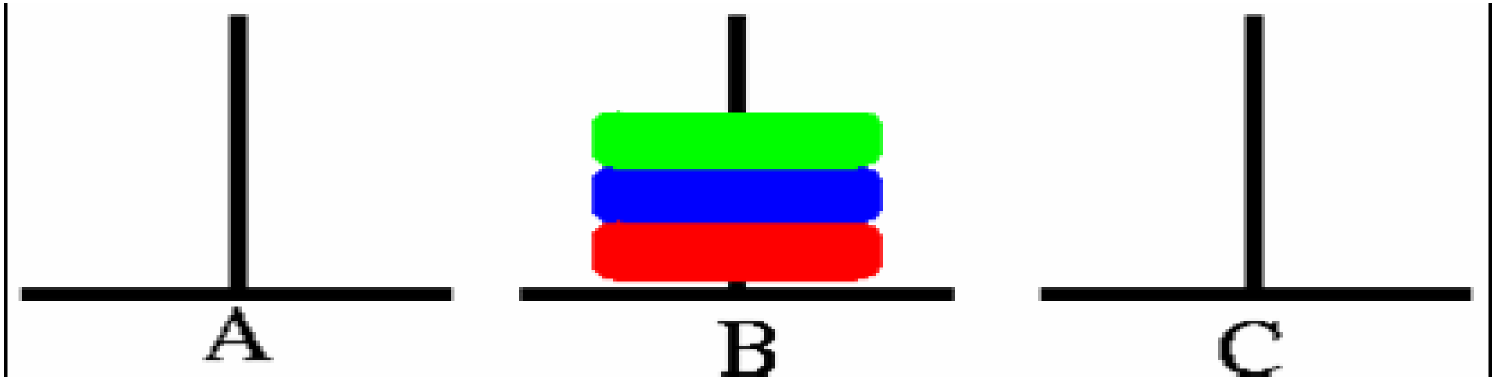
Pegs and Disks problem

- Consider the following problem. We have 3 pegs and 3 disks.
- Operators: one may move the topmost disk on any needle to the topmost position to any other needle
- In the goal state all the pegs are in the needle B as shown in the figure below.

Initial State



Goal States

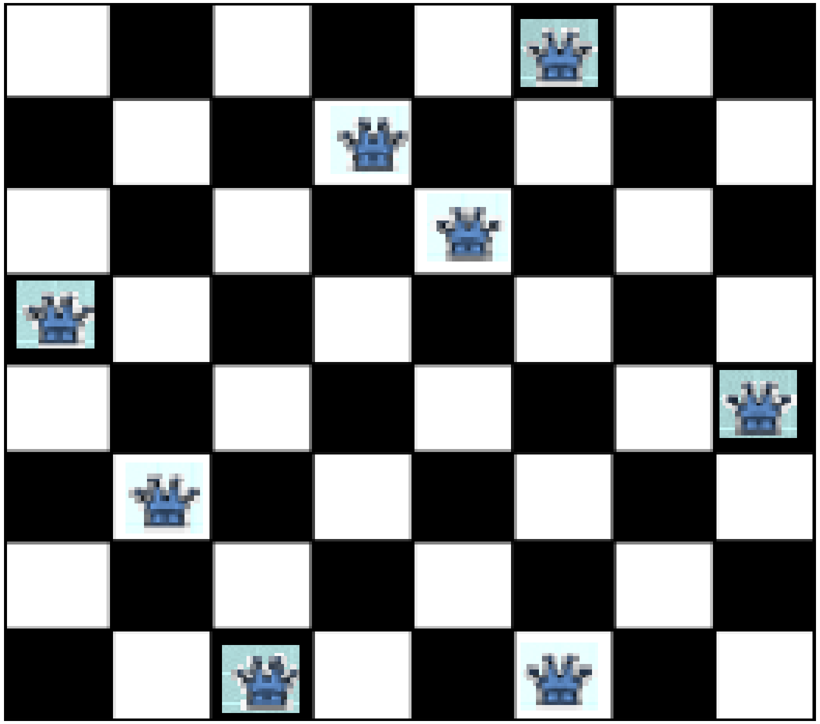


- Now we will describe a sequence of actions that can be applied on the initial state.
- Step 1: Move A → C
- Step 2: Move A → B
- Step 3: Move A → C
- Step 4: Move B→ A
- Step 5: Move C → B
- Step 6: Move A → B
- Step 7: Move C→ B

8 queens problem

- The problem is to place 8 queens on a chessboard so that no two queens are in the same row, column or diagonal

Problem space?



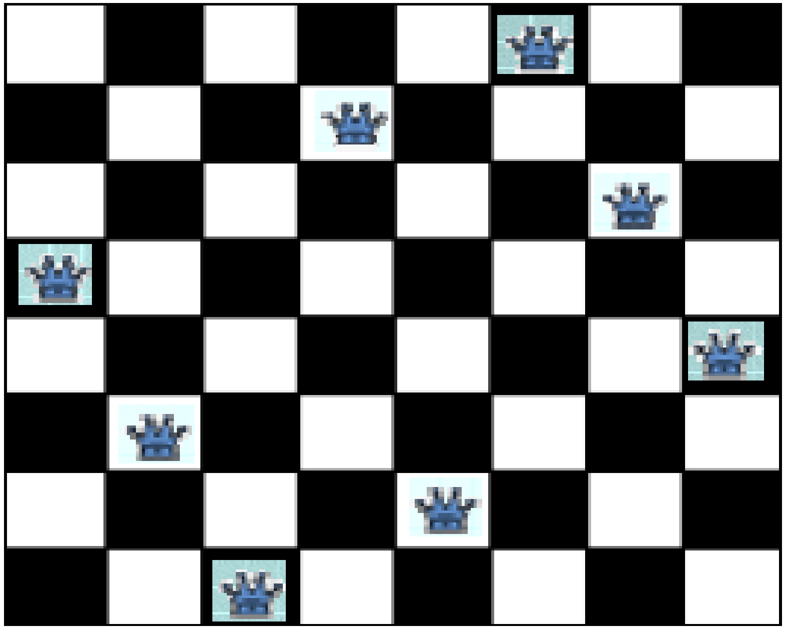
4- queens problem

	X		
			X
X			
		X	

N queens problem formulation

- States: Any arrangement of 0 to 8 queens on the board
- Initial state: 0 queens on the board
- Successor function: Add a queen in any square
- Goal test: 8 queens on the board, none are attacked

One solution



8 puzzle problem

5	4	
6	1	8
7	3	2

Initial State

1	4	7
2	5	8
3	6	

Goal State

Homework

Assignment:

1: Explain the history of AI

2: Analyse each of them and solve using AI problem solving techniques

(a) Missionaries and cannibals

(b) 8-puzzle

- A **production system** (or **production rule system**) is a computer program typically used to provide some form of artificial intelligence, which consists primarily of a set of rules about behavior. These rules, termed **productions**, are a basic representation found useful in automated planning, expert systems and action selection. A production system provides the mechanism necessary to execute productions in order to achieve some goal for the system.

- Productions consist of two parts: a sensory precondition (or "IF" statement) and an action (or "THEN").
- If a production's precondition matches the current state of the world, then the production is said to be *triggered*.
- If a production's action is executed, it is said to have *fired*.
- A production system also contains a database, sometimes called working memory, which maintains data about current state or knowledge, and a rule interpreter.
- The rule interpreter must provide a mechanism for prioritizing productions when more than one is triggered.

- A production system is a tool used in artificial intelligence and especially within the applied AI domain known as expert systems.
-
- Production systems consist of a database of rules, a working memory, a matcher, and a procedure that resolves conflicts between rules.

What is a Production System?

- A PS is a computer program typically used to provide some form of AI, which consists a set of rules about behavior.
- A PS provides the mechanism necessary to execute productions in order to achieve some goal for the system.
- Used as the basis for many rule-based expert systems

What is a Production System?

- A ***production system*** consists of four basic components:

1. A ***set of rules*** of the form *Ci ® Ai* *or*

C1, C2, … Cn => A1 A2 …Am

Left hand side (LHS) *Right hand side (RHS)*

Conditions/antecedents *Conclusion/consequence*

where *Ci* is the condition part and *Ai* is the action part.

1. The condition determines when a given rule is applied, and the action determines what happens when it is applied.
2. ***knowledge databases/ working memory*** that contain whatever information is relevant for the given problem & also maintains data about current state or knowledge. Some parts of the database may be permanent, while others may temporary and only exist during the solution of the current problem. The information in the databases may be structured in any appropriate manner.
3. A ***control strategy*** that determines the order in which the rules are applied to the database, and provides a way of resolving any conflicts that can arise when several rules match at once.
4. A ***rule applier*** which is the computational system that implements the control strategy and applies the rules.

Production rule for water jug problem

1. (x, y) → (4, y), If x < 4 fill the 4-gallon jug.
2. (x, y) → (x,3), If y < 3 fill the 3-gallon jug.
3. (x, y) → (x- d , y), If x > 0 pour some water out of the 4-gallon jug
4. (x, y) → (x, y - d), If y > 0 pour some water out of the 4-gallon jug
5. (x, y) → (0, y) If x > 0 empty the 4-gallon jug.
6. (x, y) → (x, 0), If y > 0 empty the 3-gallon jug.
7. (x, y) → (4, y – (4 – x)), if x + y >= 4 & y > 0 pour water from the 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full.
8. (x, y) → (x – (3 – y), 3), if x + y >= 4 & y > 0 pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full.

Production rule for water jug problem

9. (x, y) → (x + y, 0), if x + y <= 4 & y > 0 pour all the water from the 3-gallon jug into the 4-gallon jug.
10. (x, y) → (0, x + y), if x + y <= 3 & x > 0 pour all the water from the 4-gallon jug into the 3-gallon jug.

11. (0, 2) → (2, 0), pour 2-g from 3-g to 4-g
12. (2, y) → (0, y)

One solution of water jug problem

Rule applied	4-Gallon	3-Gallon
Initial state	0	0
Rule 2	0	3
Rule 9	3	0
Rule 2	3	3
Rule 7	4	2
Rule 5 or 12	0	2
Rule 9 or 11	2	0

Problem of Conflict Resolution

- When there are more then one rule that can be fired in a situation and the rule interpreter can not be decide which is to be fired, what is the order of triggering and whether to apply it .

Some Resolution Strategies

- Perform the first. the system chooses the first rule that matches.
- Sequencing techniques. adopt the rules in the sequence they are.
- Perform the most specific. if there are two matching rules and one rule is more specific than the other, activate the most specific.
- Most recent policy. chooses newly added rule.

search

- Search → process of locating a solution to a problem by any method in a search tree or search space until a goal node is found.
- Search Space → A set of possible permutation that can be examined by any search method in order to find solution.
- Search Tree → A tree that is used to represent a search problem and is examined by search method to search for a solution.

To do a search process the following are needed :--

The initial state description.

A set of legal operators.

The final or goal state.

Search Tree – Terminology

- Root Node: The node from which the search starts.
- Leaf Node: A node in the search tree having no children.
- Ancestor/Descendant: X is an ancestor of Y is either X is Y's parent or X is an ancestor of the parent of Y. If S is an ancestor of Y, Y is said to be a descendant of X.
- Branching factor: the maximum number of children of a non-leaf node in the search tree
- Path: A path in the search tree is a complete path if it begins with the start node and ends with a goal node. Otherwise it is a partial path.
- We also need to introduce some data structures that will be used in the search algorithms.

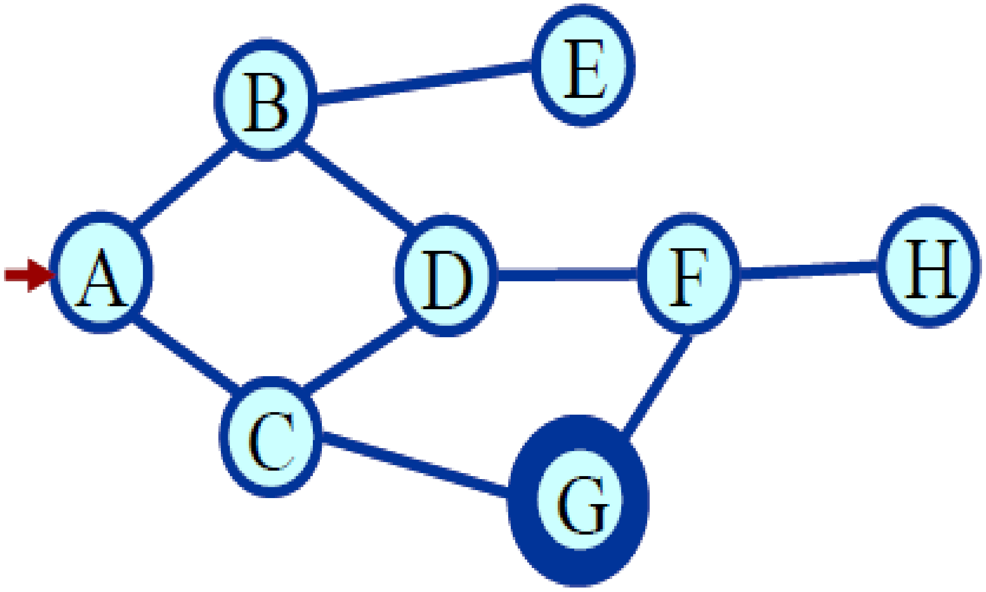


Figure 1: A State Space Graph

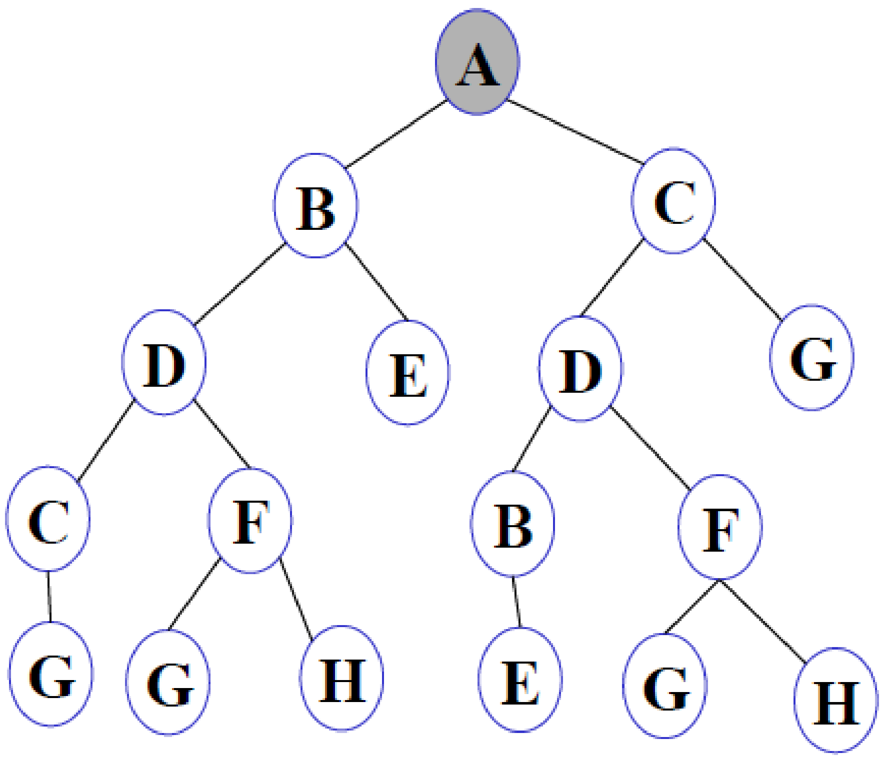


Figure 2: Search tree for the state space graph in Figure 25

Evaluating Search strategies

- We will look at various search strategies and evaluate their problem solving performance. What are the characteristics of the different search algorithms and what is their efficiency? We will look at the following three factors to measure this.

Search Strategy Evaluation

Completeness: We will say a search method is "complete" if it has both the following properties:

- if a goal exists then the search will always find it
- if no goal exists then the search will eventually finish and be able to say that no goal exists

Time complexity: how long does it take?(number of nodes expanded)

Space complexity: how much memory is needed?

Optimality: is a high-quality solution found? Does the solution have low cost or the minimal cost? What is the search cost associated with the time and memory required to find a solution?

- **Which path to find?**
- The objective of a search problem is to find a path from the initial state to a goal state. If there are several paths which path should be chosen? Our objective could be to find any path, or we may need to find the shortest path or least cost path.

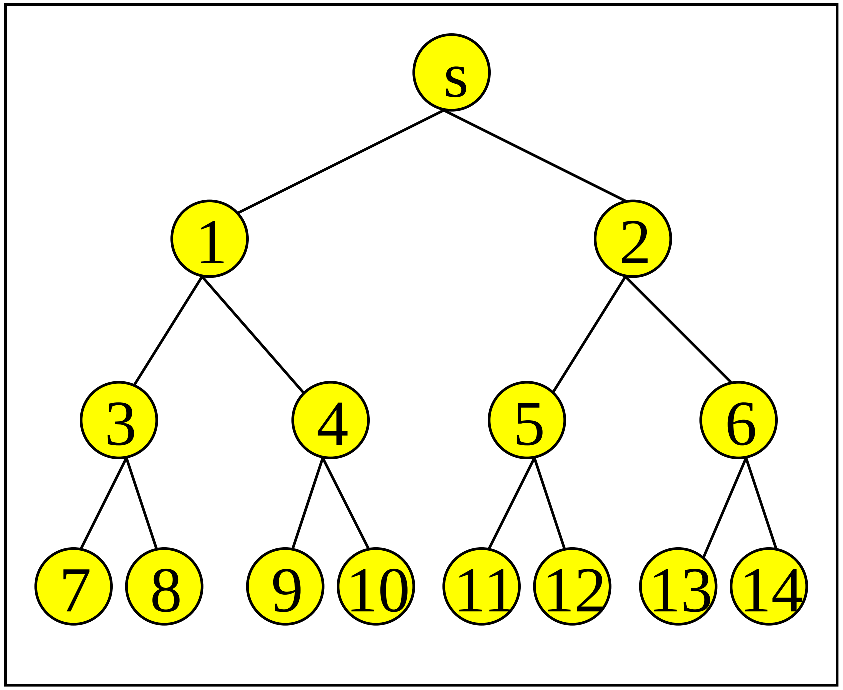
- The different search strategies that we will consider include the following:
- 1. Blind Search strategies or Uninformed search
  - a. Depth first search
  - b. Breadth first search
  - c. Iterative deepening search
  - d. Iterative broadening search
- 2. Informed Search
- 3. Constraint Satisfaction Search
- 4. Adversary Search

Types of Search

- Uninformed or blind or Brute force search
  - No information about the number of steps
  - No information about the path cost
  - blind search or uninformed search that does not use any extra information about the problem domain.
- Informed or heuristic search
  - Information about possible path costs or number of steps is used

Uninformed Search
Breadth-first search

- Root node is expanded first
- All nodes at depth *d* in the search tree are expanded before the nodes at depth *d+1*
- Implemented by putting all the newly generated nodes at the end of the queue



Breadth first search queues

Loopno	*nodes*	*expanded*
0	[s]	∅
1	[1 2]	[s]
2	[2 3 4]	[1 s]
3	[3 4 5 6]	[2 1 s]
4	[4 5 6 7 8]	[3 2 1 s]
5	[5 6 7 8 9 10]	[4 3 2 1 s]
6	[6 7 8 9 10 11 12]	[5 4 3 2 1 s]
:	:	:

Algorithm of BFS

Step 1: put the initial node on a list S.

Step 2 : if (S is empty) or (S = goal) terminate search.

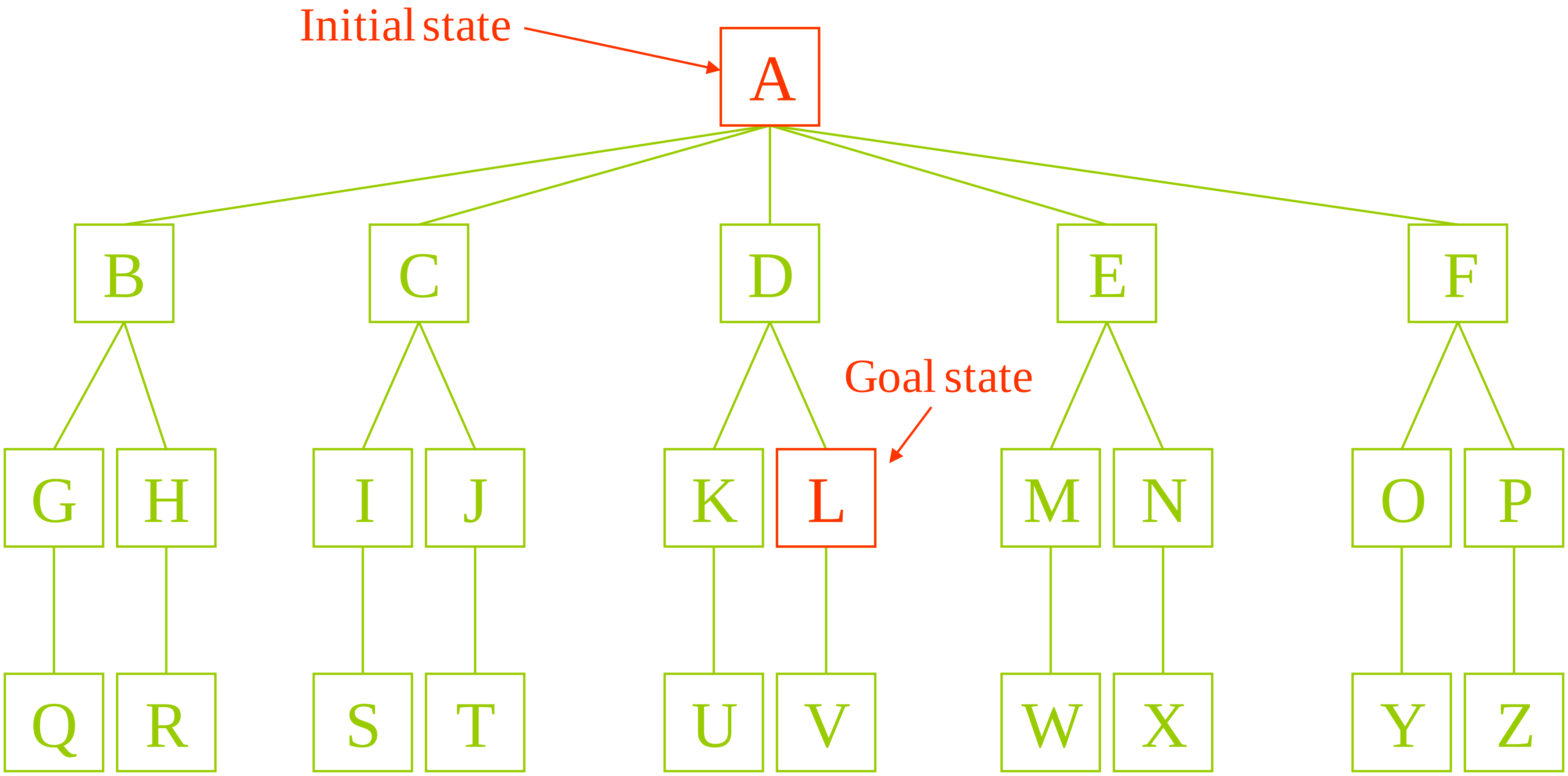
Step 3 : remove the first node from S. call this node a.

Step 4 : if (a = goal) terminate search with success.

Step 5 :Else if node a has successor, generate all of them and add them at the tail of S.

Step 6 : go to to step 2.

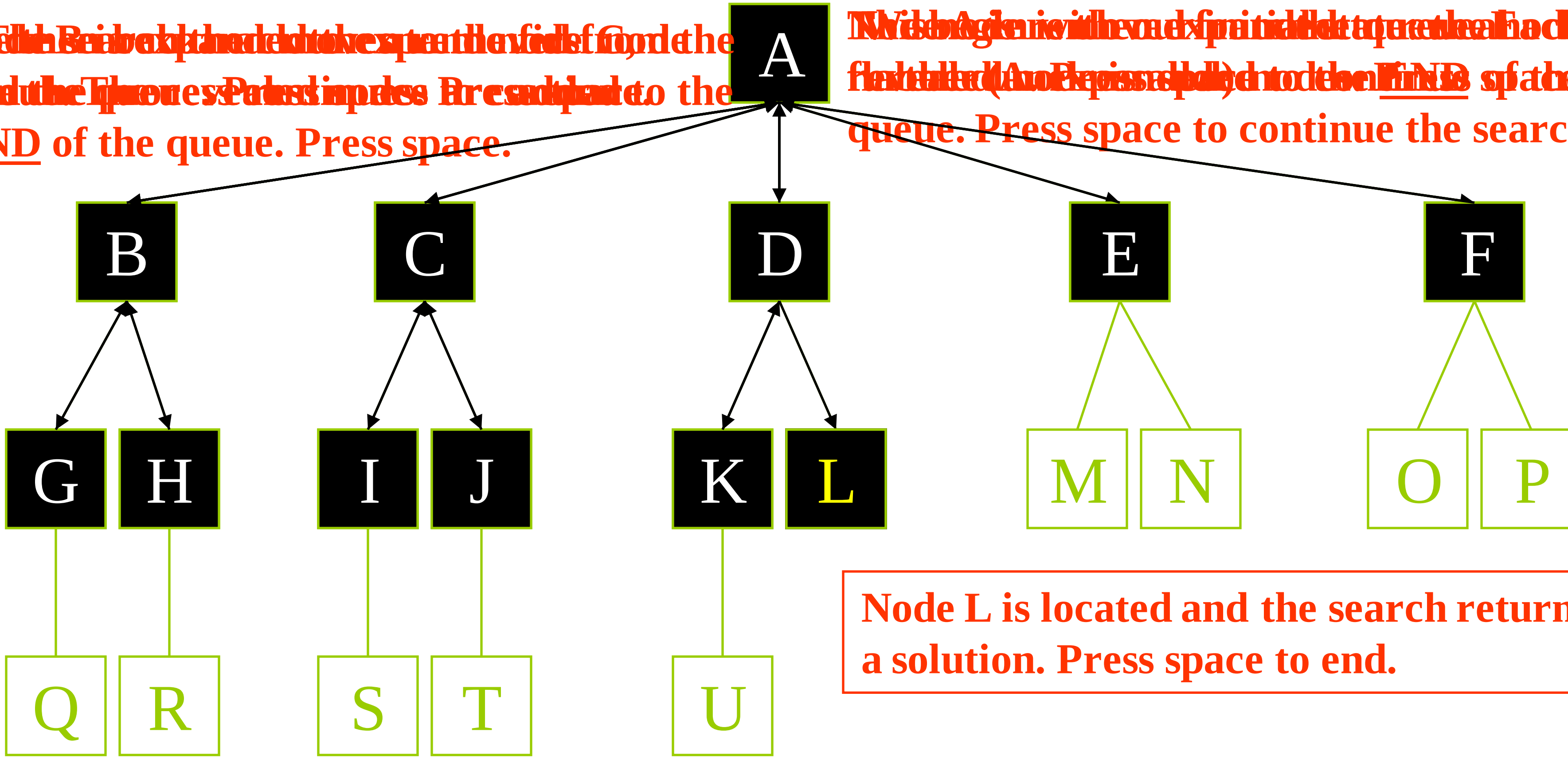
The example node set



Press space to see a BFS of the example node set

END of the queue. Press space.

queue. Press space to continue the search.



Node L is located and the search returns a solution. Press space to end.

Press space to begin the search

Size of Queue: 0	**Queue: Empty**

Nodes expanded: 11	**FINISHED SEARCH**	**Current level: 2**

BREADTH-FIRST SEARCH PATTERN

Time Complexity :

$$1 + b + b^2 + b^3 + \ldots + \ldots\ldots b^{d.}$$

Hence Time complexity = O (b^d)

Space Complexity :

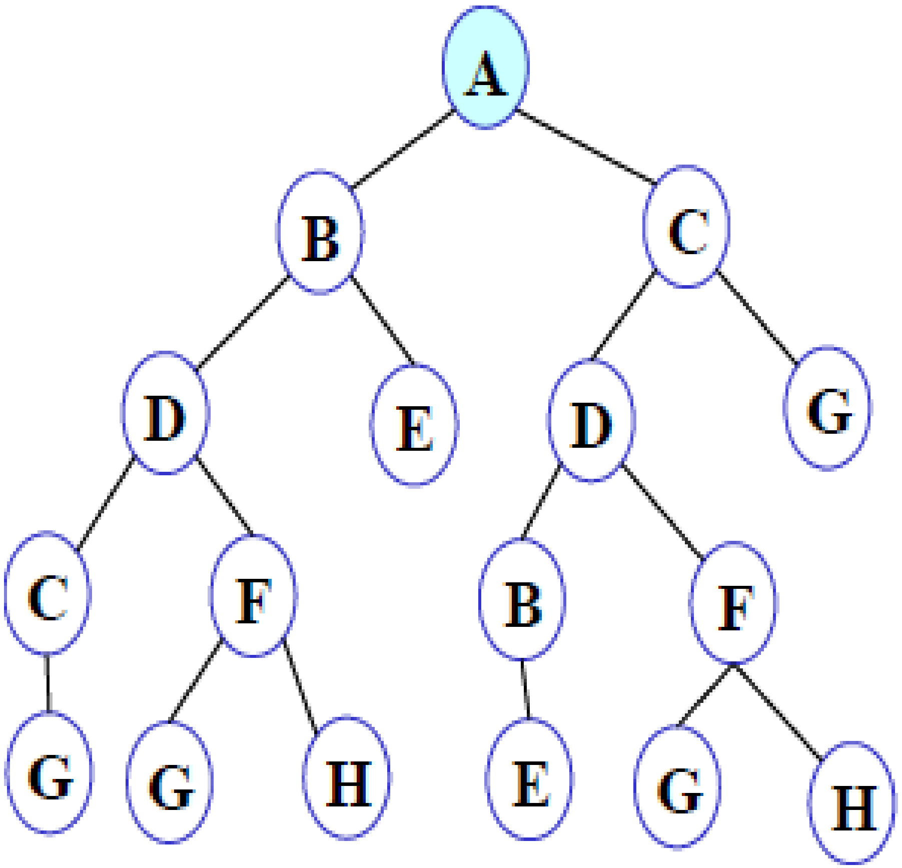
$$1 + b + b^2 + b^3 + \ldots + \ldots\ldots b^{d.}$$

Hence Time complexity = O (b^d)

Uninformed Search
Breadth-first search

- Breadth-first search merits
  - *Complete:* If there is a solution, it will be found
  - *Optimal*: Finds the nearest goal state
- Breadth-first search problem:
- Time complexity
- Memory intensive
- Remembers all unwanted nodes

show how breadth first search works on this graph.

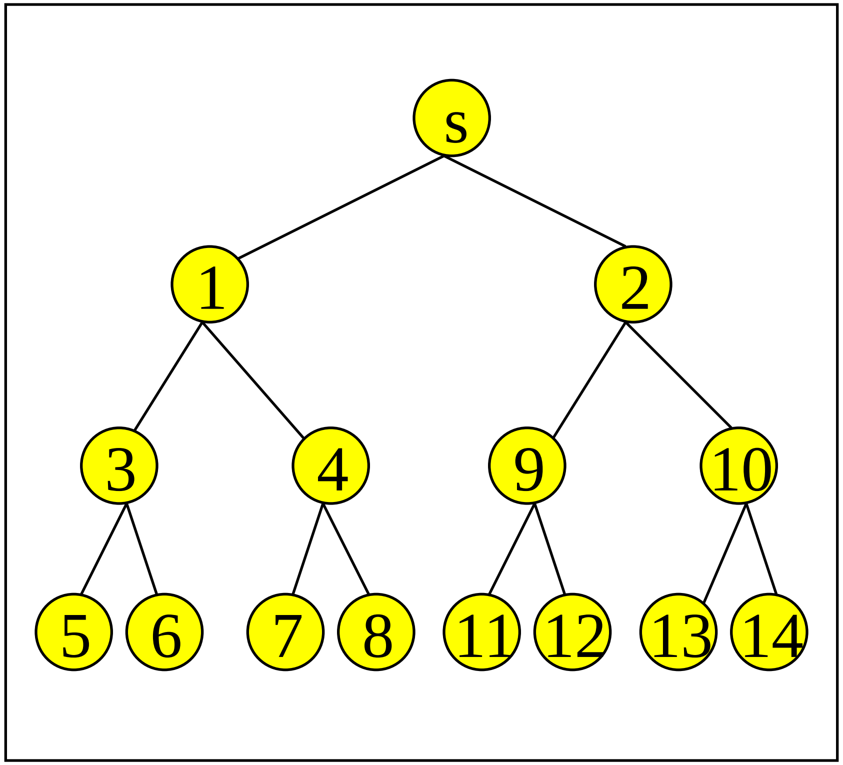


- Breadth first search is:
- Complete. :→ The algorithm is optimal (i.e., admissible) if all operators have the same cost. Otherwise, breadth first search finds a solution with the shortest path length.
- The algorithm has exponential time and space complexity. Suppose the search tree can be modeled as a b-ary tree as shown in Figure 3. Then the time and space complexity of the algorithm is O(bd) where d is the depth of the solution and b is the branching factor (i.e., number of children) at each node.
- A complete search tree of depth d where each non-leaf node has b children, has a total of **$1 + b + b^2 + ... + b^d = (b^{(d+1)} - 1)/(b-1)$ nodes**

- Consider a complete search tree of depth 15, where every node at depths 0 to14 has 10 children and every node at depth 15 is a leaf node. The complete search tree in this case will have $O(10^{15})$ nodes. If BFS expands 10000 nodes per second and each node uses 100 bytes of storage, then BFS will take 3500 years to run in the worst case, and it will use 11100 terabytes of memory. So you can see that the breadth first search algorithm cannot be effectively used unless the search space is quite small. You may also observe that even if you have all the time at your disposal, the search algorithm cannot run because it will run out of memory very soon.

Uninformed Search
Depth-first search

- Always expands one of the node at the deepest level of the tree
- Only returns when the search hits a dead end
- Implemented by putting the newly generated nodes at the front of the queue



Depth first search queues

Loopno	*nodes*	*expanded*
0	[s]	∅
1	[1 2]	[s]
2	[3 4 2]	[1 s]
3	[5 6 4 2]	[3 1 s]
4	[6 4 2]	[5 3 1 s]
5	[4 2]	[6 5 3 1 s]
6	[7 8 2]	[4 6 5 3 1 s]
:	:	:

Algorithm of DFS

Step 1: put the initial node on a list S.

Step 2 : if (S is empty) or (S = goal) terminate search.

Step 3 : remove the first node from S. call this node a.

Step 4 : if (a = goal) terminate search with success.

Step 5 :Else if node a has successor, generate all of them and add them at the beginning of S.

Step 6 : go to to step 2.

Time Complexity :

$$1 + b + b^2 + b^3 + \ldots + \ldots\ldots b^{d.}$$

Hence Time complexity = O (b^d)

Space Complexity :

Hence Time complexity = O (d)

Uninformed Search
Depth-first search

- Depth-first search merits
  - *Modest memory requirements*: only the current path from the root to the leaf node needs to be stored.
  - *Time complexity*
    - With many solutions, depth-first search is often faster than breadth-first search, but the worst case is still O (b^m)

Properties of Depth First Search

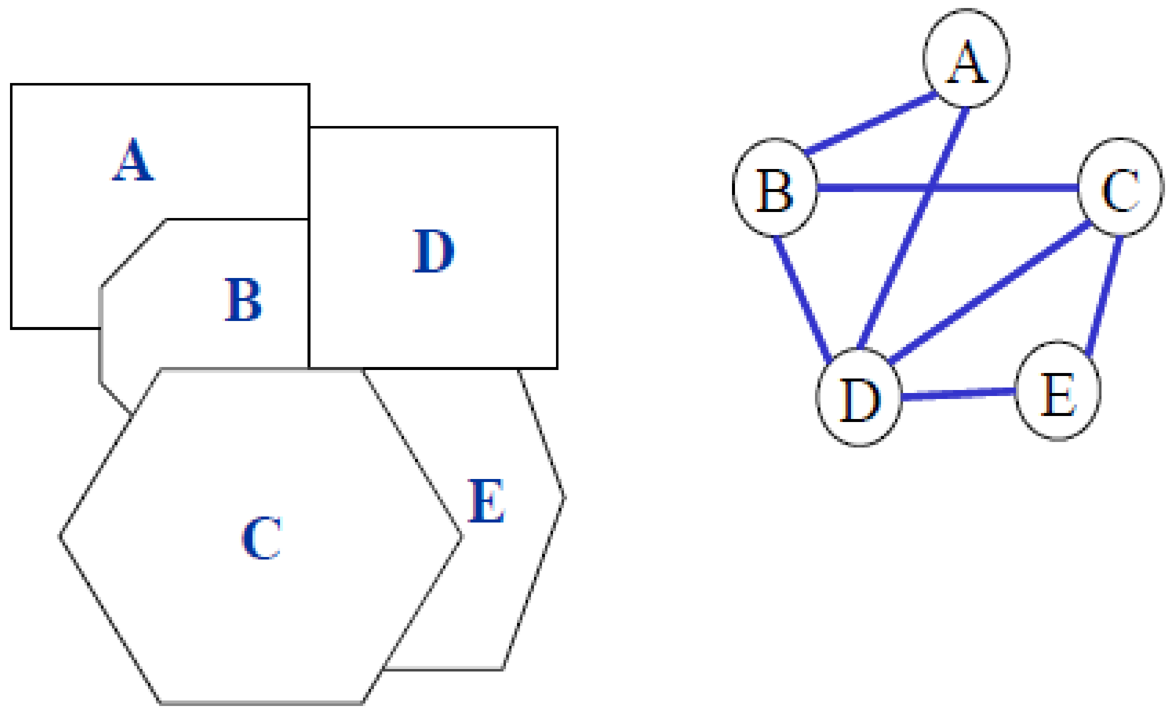
- Let us now examine some properties of the DFS algorithm. The algorithm takes exponential time. If N is the maximum depth of a node in the search space, in the worst case the algorithm will take time $O(b^d)$. However the space taken is linear in the depth of the search tree, O(bN).
- Note that the time taken by the algorithm is related to the maximum depth of the search tree. If the search tree has infinite depth, the algorithm may not terminate. This can happen if the search space is infinite. It can also happen if the search space contains cycles. The latter case can be handled by checking for cycles in the algorithm. Thus Depth First Search is not complete.

Questions

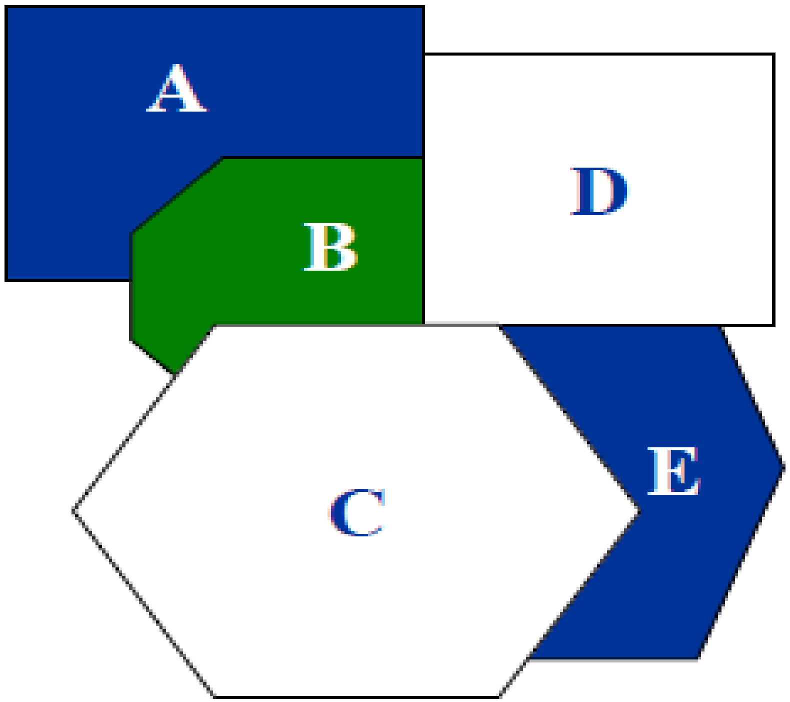
- Give the initial state, goal test, sucessor function, and cost function for each of the following. Choose a formulation that is precise enough to be implemented.
- a) You have to colour a planar map using only four colours, in such a way that no two adjacent regions have the same colour.

Solutions

- The map is represented by a graph. Each region corresponds to a vertex of the graph. If two regions are adjacent, there is an edge connecting the corresponding vertices.
- The vertices are named <v1, v2, … , vN>.
- The colors are represented by c1, c2, c3, c4.
- A state is represented as a N-tuple representing the colors of the vertices. A vertex has color x if its color has not yet been assigned. An example state is:
- {c1, x, c1, c3, x, x, x …}
- color(i) denotes the color of si.
- Consider the map below consisting of 5 regions namely A, B, C, D and E. The adjacency information is represented by the corresponding graph shown.

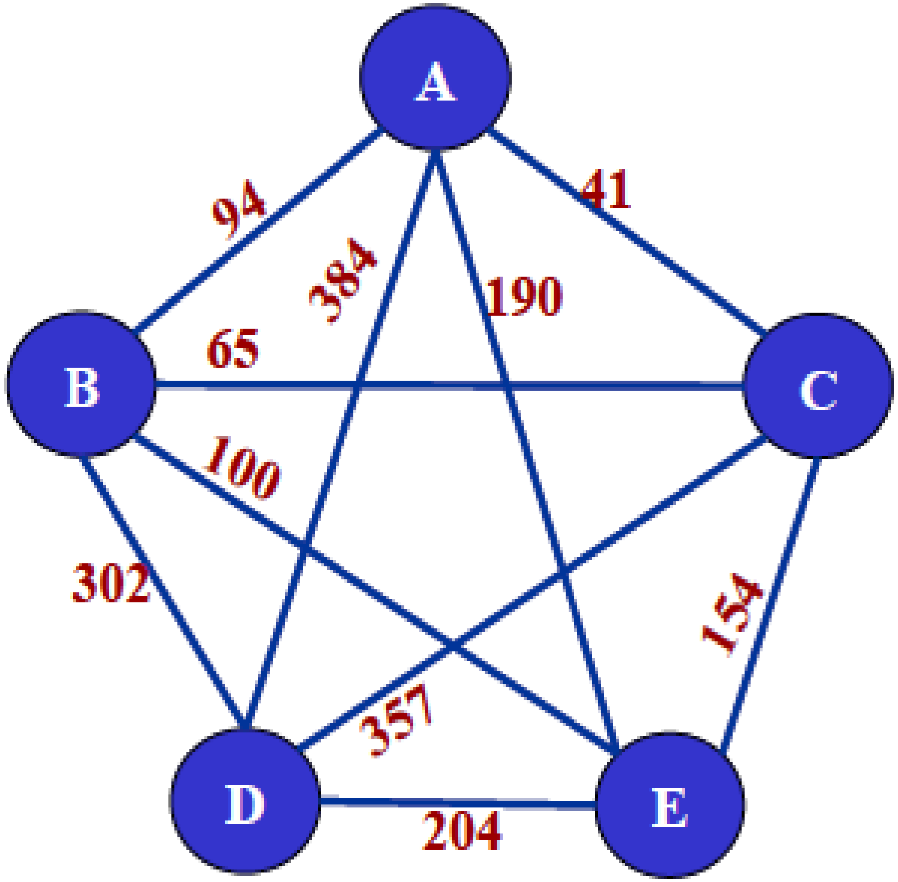

A
B
D
C
E
A
B
C
D
E

A state of this problem is shown below.



- This state is represented as {blue, green, x, x, blue}.
- The initial state for this problem is given as {x, x, x, x, x}
- The goal test is as follows. For every pair of states s^i and s^j that are adjacent, colour(i) must be different from colour(j).
- The successor functions are of the form:
- • Change (i, c): Change the colour of a state i to c.

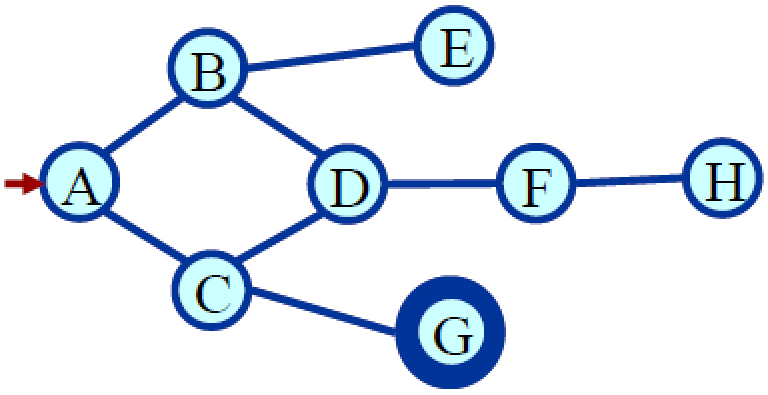
2. In the travelling salesperson problem (TSP) there is a map involving N cities some of which are connected by roads. The aim is to find the shortest tour that starts from a city, visits all the cities exactly once and comes back to the starting city.


A
B
C
D
E
94
41
384
190
65
100
302
154
357
204

- Y: set of N cities
- d(x,y) : distance between cities x and y. $x,y \in Y$
- A state is a Hamiltonian path (does not visit any city twice)
- X: set of states
- X: set of states. X = {(x**1, x2, …, xn)|** n=1, …, N+1, xi ∈Y for all I,
- xi ≠ xj unless i=1, j=N+1}
- Successors of state (x**1, x2, …, xn):**
- δ (x**1, x2, …, xn) = {(x1, x2, …, xn, xn+1) | xn+1∈ Y**
- **xn+1 ≠ xi for all 1≤ i≤ n }**
- The set of goal states include all states of length N+1

- **Missionaries & Cannibals problem: 3 missionaries & 3 cannibals are on one side of the river. 1 boat carries 2. Missionaries must never be outnumbered by cannibals. Give a plan for all to cross the river. State: <M, C, B>**
- M: no of missionaries on the left bank
- C: no of cannibals on the left bank
- B: position of the boat: L or R
- Initial state: <3, 3, L>
- Goal state: <0, 0, R>
- Operators: <M,C> ► M: No of missionaries on the boat
- ► C: No of cannibals on the boat
- Valid operators: <1,0> <2,0>, <1,1>, <0,1> <0,2>

Starting from state A, execute DFS. The goal node is G. Show the order in which the nodes are expanded. Assume that the alphabetically smaller node is expanded first to break ties.



Step	Fringe	Node Expanded	Comments
1	A		
2	B C	A	
3	D E C	B	
4	F E C	D	
5	H E C	F	
6	E C	H	
7	C	E	
8	D G	C	
9	F G	D	
10	H G	F	
11	G	H	
12		G	Goal reached!

Iterative Deepening Search

Depth-First Iterative Deepening (DFID)

- First do DFS to depth 0 (i.e., treat start node as having no successors), then, if no solution found, do DFS to depth 1, etc. DFID
- *until solution found do*
- *DFS with depth cutoff c*
- *c = c+1*

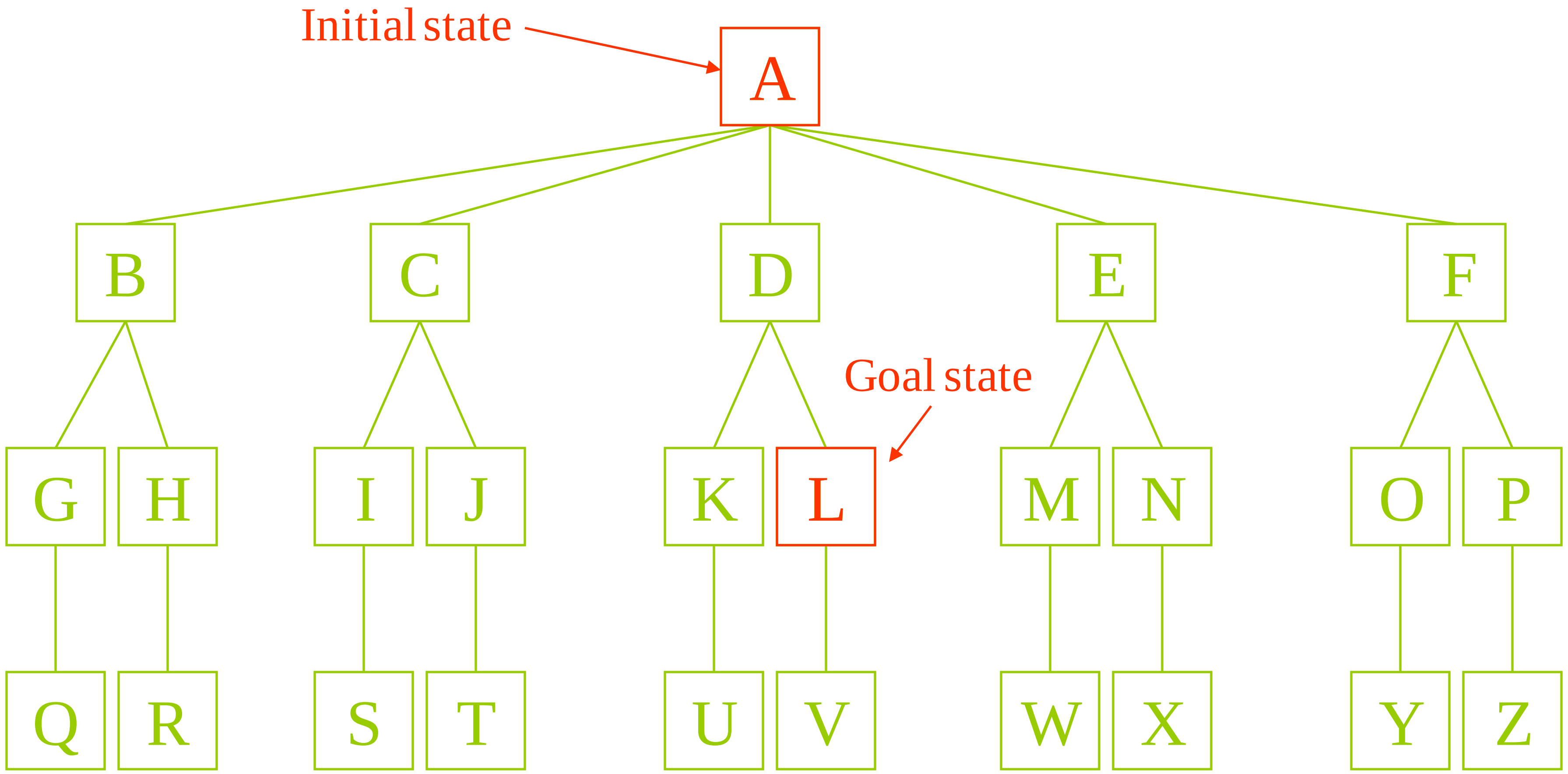
Advantage

- Linear memory requirements of depth-first search
- Guarantee for goal node of minimal depth

Iterative deepening search

- The problem with depth-limited search is deciding on a suitable depth parameter. To avoid this problem there is another search called iterative deepening search (IDS).
- This search method tries all possible depth limits; first 0, then 1, then 2 etc., until a solution is found.
- IDS may seem wasteful as it is expanding nodes multiple times. But the overhead is small in comparison to the growth of an exponential search tree
- For large search spaces where is the depth of the solution is not known IDS is normally the preferred search method.
- The following slide illustrates an iterative deepening search of 26 nodes (states) with an initial state of node A and a goal state of node L. Press space to see the example node set.

The example node set



Press space to see a IDS of the example node set

A

We begin with our initial state: the node labeled A. This node is added to the queue. Press space to continue

Node A is then expanded and removed from the queue. Press space to continue

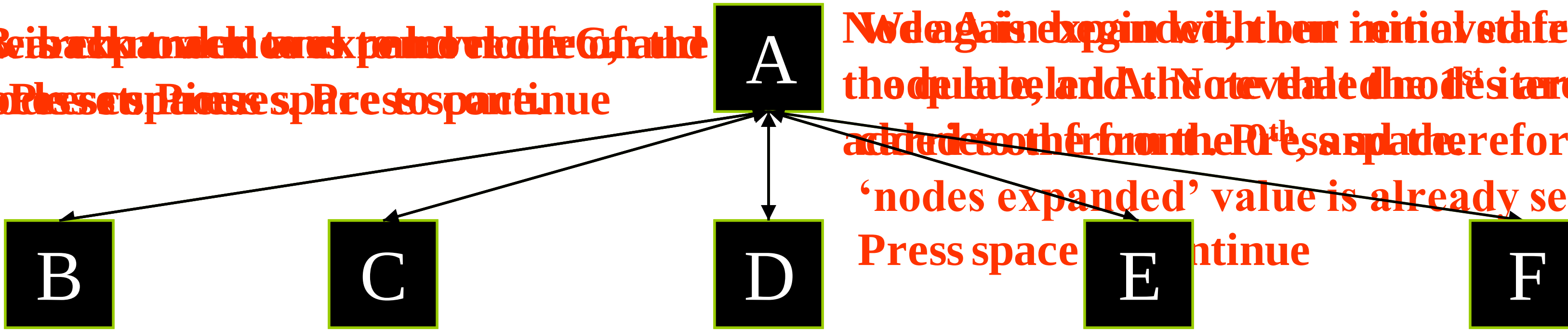
As this is the 0th iteration of the search, we cannot search past any level greater than zero. This iteration now ends, and we begin the 1st iteration.

Press space to begin the search

Size of Queue: 0	**Queue: Empty**	
Nodes expanded: 1	**Current Action: Expanding**	**Current level: 0**

ITERATIVE DEEPENING SEARCH PATTERN (0th ITERATION)

[illegible]



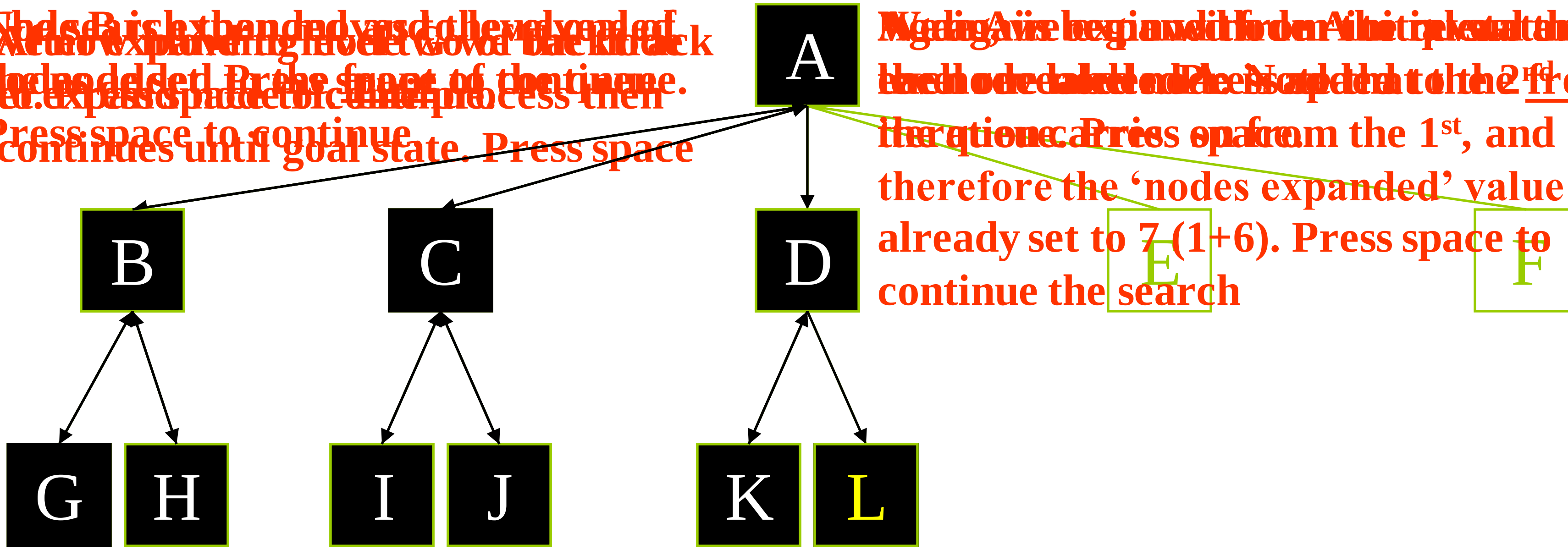
We again begin with our initial state: the node labeled A. Note that the 1st iteration carries on from the 0th, and therefore the 'nodes expanded' value is already set to 1. Press space to continue

As this is the 1st iteration of the search, we cannot search past any level greater than level one. This iteration now ends, and we begin a 2nd iteration.

Press space to begin the search

Size of Queue: 0	Queue: Empty	
Nodes expanded: 7	**Current Action: Expanding**	**Current level: 1**

ITERATIVE DEEPENING SEARCH PATTERN (1st ITERATION)



therefore the 'nodes expanded' value is already set to 7 (1+6). Press space to continue the search

Node L is located on the second level and the search returns a solution on its second iteration. Press space to end.

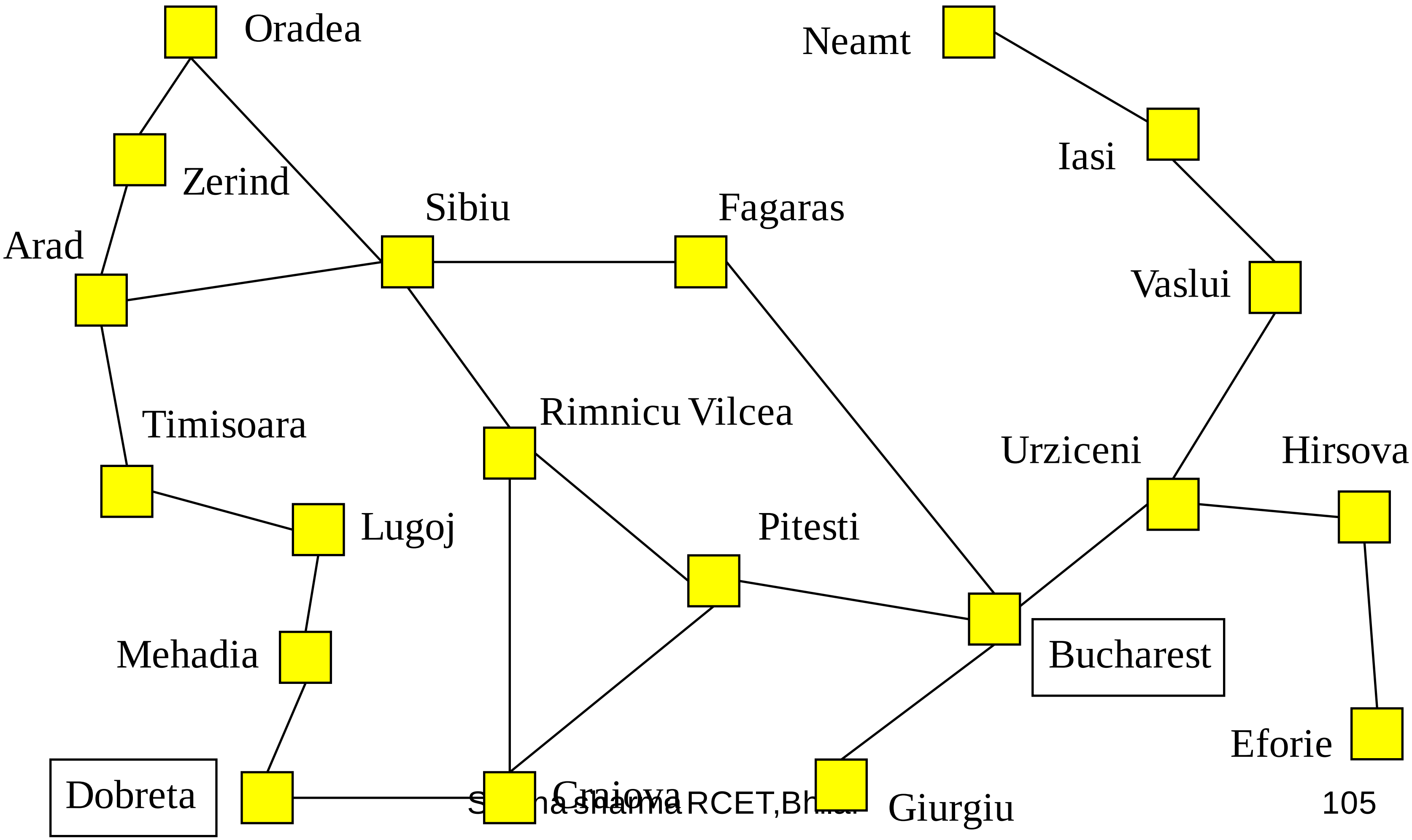
Press space to continue the search

Size of Queue: 0	**Queue: Empty**	
Nodes expanded: 16	**SEARCH FINISHED**	**Current level: 2**

ITERATIVE DEEPENING SEARCH PATTERN (2nd ITERATION)

Iterative Deepening (become deeper) Search

Go from Dobreta to Bucharest



Uninformed Search
Iterative deepening search (3)

- Iterative deepening search seems pretty dumb
  - Many states are expanded multiple times
- For most problems the overhead is small!
  - Almost all of the nodes are at the bottom level
- Search with branching factor *b* and depth *d*
  - *Depth-limited search*
    - $1+b+b^2+b^3+\ldots+b^{d-2}+b^{d-1}+b^d$
    - b = 10 and d = 5: 111,111 expansions
  - *Iterative deepening search*
    - $(d+1)1+(d)b+(d-1)b^2+\ldots+3b^{d-2}+2b^{d-1}+1b^d$
    - b = 10 and d = 5: 123,456 expansions

➢ Only about 11% worse

Uninformed Search
Iterative deepening search

- How to choose the maximum depth limit?
  - *Rumania example*: the *diameter* of the state space is 9 instead of 19
  - In most problems a good depth limit is unknown
- Iterative deepening search
  - Try all possible depths
  - *Optimal*, *complete* like breadth-first search and has *modest memory requirements* like depth-first search

function Iterative-Deepening-Search(*problem*) **returns** a solution sequence
  inputs: *problem*
  for *depth* $\leftarrow$ 0 **to** ∞ **do**
    if Depth-Limited-Search(*problem*, *depth*) succeeds
    then return its result
  end
  return failure

Heuristic search or Informed search

Informed Search Methods

- General-search algorithm
  - Knowledge can only be applied in the queuing function
- Informed search
  - Use knowledge about the expected distance to the goal state
  - This knowledge is provided by an *evaluation function*
  - Returns the desirability of expanding the node

- We have seen that uninformed search methods that systematically explore the state space and find the goal. They are inefficient in most cases. Informed search methods use problem specific knowledge, and may be more efficient. At the heart of such algorithms there is the concept of a heuristic function.

- Heuristic means “rule of thumb”. To quote Judea Pearl, “Heuristics are criteria, methods or principles for deciding which among several alternative courses of action promises to be the most effective in order to achieve some goal”. In heuristic search or informed search, heuristics are used to identify the most promising search path.

"It is defined as a method that provide a better guess about the correct choice to make at any junction that would be achieved by random guessing." OR

"It is defined as a method or as a rule or as a trick. it is a piece of information that is used to make search or another problem solving method, more effective and more efficient."

A heuristic is a method that

- Might not always find the best solution.
- But is guaranteed to find a good solution in reasonable time.
- Heuristics are approximation used to minimize the search process
- Useful in solving tough problems which

  -- could not be solved any other way.

  -- solutions take an infinite time or very long time to compute.

- Heuristic function : a function that estimate the value of a state, It is an approximation used to minimize the search process .

- Heuristic Knowledge : knowledge of approaches that are likely to work or of properties that are likely to be true (but not guaranteed).

Example of Heuristic Function

- A heuristic function at a node n is an estimate of the optimum cost from the current node to a goal. It is denoted by *h(n).*

h(n) = estimated cost of the cheapest path from node n to a goal node

Example 1: We want a path from Kolkata to Guwahati

- Heuristic for Guwahati may be straight-line distance between Kolkata and Guwahati
- *h(Kolkata) = euclideanDistance(Kolkata, Guwahati)*

Example 2: 8-puzzle: Misplaced Tiles Heuristics is the number of tiles out of place.

2	8	3
1	6	4
	7	5

Initial State

1	2	3
8		4
7	6	5

Goal state

Figure 1: 8 puzzle

- The first picture shows the current state *n, and the second picture the goal state.*
- $h(n) = 5$
- because the tiles 2, 8, 1, 6 and 7 are out of place.
- Manhattan Distance Heuristic: Another heuristic for 8-puzzle is the Manhattan distance heuristic. This heuristic sums the distance that the tiles are out of place. The distance of a tile is measured by the sum of the differences in the x-positions and the y-positions.
- For the above example, using the Manhattan distance heuristic,
- $h(n) = 1 + 1 + 0 + 0 + 0 + 1 + 1 + 1 + 1 = 6$

Hill Climbing Algorithm

- Hill climbing is a graph search algorithm where the current path is extended with a successor node which is closer to the solution than the end of the current path.
- In simple hill climbing, the first closer node is chosen whereas in steepest ascent hill climbing all successors are compared and the closest to the solution is chosen.
- Both forms fail if there is no closer node. This may happen if there are local maxima in the search space which are not solutions. Steepest ascent hill climbing is similar to best first search but the latter tries all possible extensions of the current path in order whereas steepest ascent only tries one.
- Hill climbing is sometimes called greedy local search because it grabs a good neighbor state without thinking ahead about where to go next.
- Hill climbing often makes very rapid progress towards a solution, because it is usually quite easy to improve a bad state. Unfortunately, hill climbing often gets stuck for the following reasons:

- 1. Local Maxima:
- A local maximum is a peak that is higher than each of its neighboring states, but lower than the global maximum. Hill-climbing algorithms that reach the vicinity of a local maximum will be drawn upwards towards the peak, but will then be stuck with nowhere else to go.
- 2. Ridges:
- Ridges result in a sequence of local maxima that is very difficult
- for greedy algorithms to navigate.

3. Plateaux:

- A plateau is an area of the state space landscape where the evaluation function is flat. It can be a flat local maximum, from which no uphill exit exists, or from which it is possible to make progress.
- • Hill climbing operate on complete-state formulations, keeping only a small number of nodes in memory
- Hill climbing is used widely in artificial intelligence fields, for reaching a goal state from a starting node. Choice of next node/ starting node can be varied to give a list of related algorithms.
- • The problem with hill climbing is that it may find only local maxima. Unless the heuristic is good / smooth, it doesn't reach global maxima.

Hill-climbing Search

- Generate nearby successor states to the current state based on some knowledge of the problem.
- Pick the best of the bunch and replace the current state with that one.
- Loop (until?)

Hill-Climbing Search

```
function HILL-CLIMBING(problem) return a state that is a local
    maximum
    input: problem, a problem
    local variables: current, a node.
                     neighbor, a node.

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest valued successor of current
        if VALUE [neighbor] ≤ VALUE[current] then return
                 STATE[current]
        current ← neighbor
```

Hill-climbing

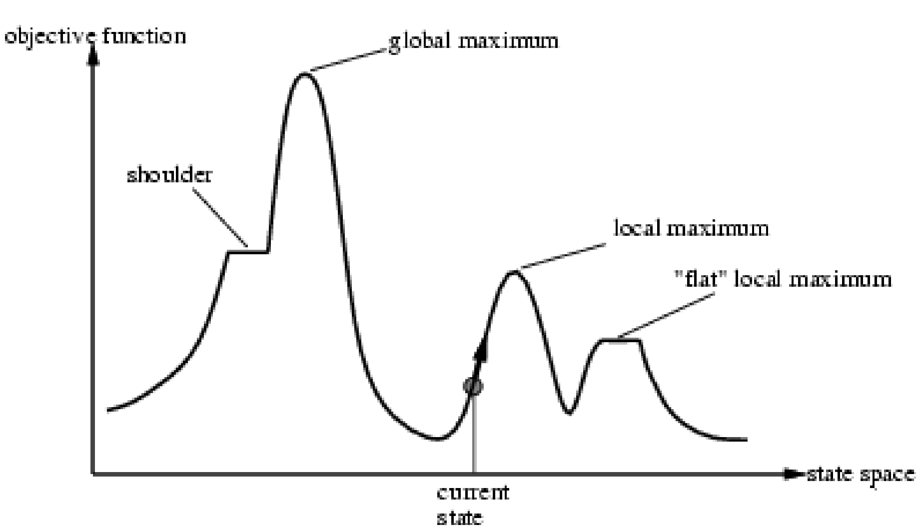
- Implicit in this scheme is the notion of a *neighborhood* that in some way preserves the cost behavior of the solution space…
  - Think about the TSP problem again
  - If I have a current tour what would a neighboring tour look like?
    - This is a way of asking for a successor function.

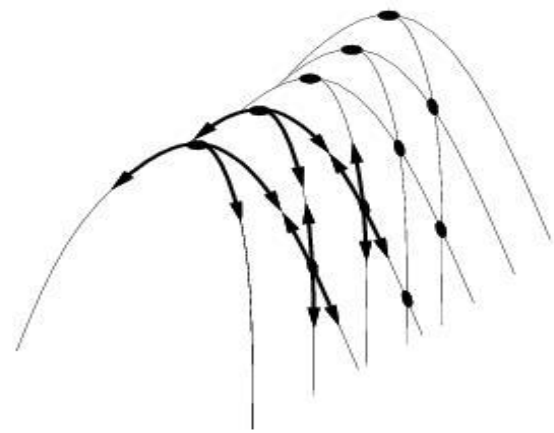
Hill-climbing Search

- The successor function is where the intelligence lies in hill-climbing search
- It has to be conservative enough to preserve significant “good” portions of the current solution
- And liberal enough to allow the state space to be preserved without degenerating into a random walk

Hill-climbing search

- Problem: depending on initial state, can get stuck in various ways
-





Local Maxima (Minima)

- Hill-climbing is subject to getting stuck in a variety of local conditions…
- Two solutions
  – Random restart hill-climbing
  – Simulated annealing

Random Restart Hillclimbing

- Pretty obvious what this is….
  - Generate a random start state
  - Run hill-climbing and store answer
  - Iterate, keeping the current best answer as you go
  - Stopping… when?
- Give me an optimality proof for it.

Annealing

- Based on a metallurgical metaphor
  - Start with a temperature set very high and slowly reduce it.
  - Run hillclimbing with the twist that you can occasionally replace the current state with a worse state based on the current temperature and how much worse the new state is.

Annealing

- More formally…
  - Generate a new neighbor from current state.
  - If it's better take it.
  - If it's worse then take it with some probability proportional to the temperature and the delta between the new and old states.

Simulated annealing

```
function SIMULATED-ANNEALING( problem, schedule) return a solution state
    input: problem, a problem
           schedule, a mapping from time to temperature
    local variables: current, a node.
                     next, a node.
                     T, a “temperature” controlling the probability of downward steps

    current ← MAKE-NODE(INITIAL-STATE[problem])
    for t ← 1 to ∞  do
           T ← schedule[t]
           if T = 0 then return current
           next ← a randomly selected successor of current
           ΔE ← VALUE[next] - VALUE[current]
           if ΔE > 0 then current ← next
           else current ← next only with probability e^(ΔE /T)
```

Properties of simulated annealing search

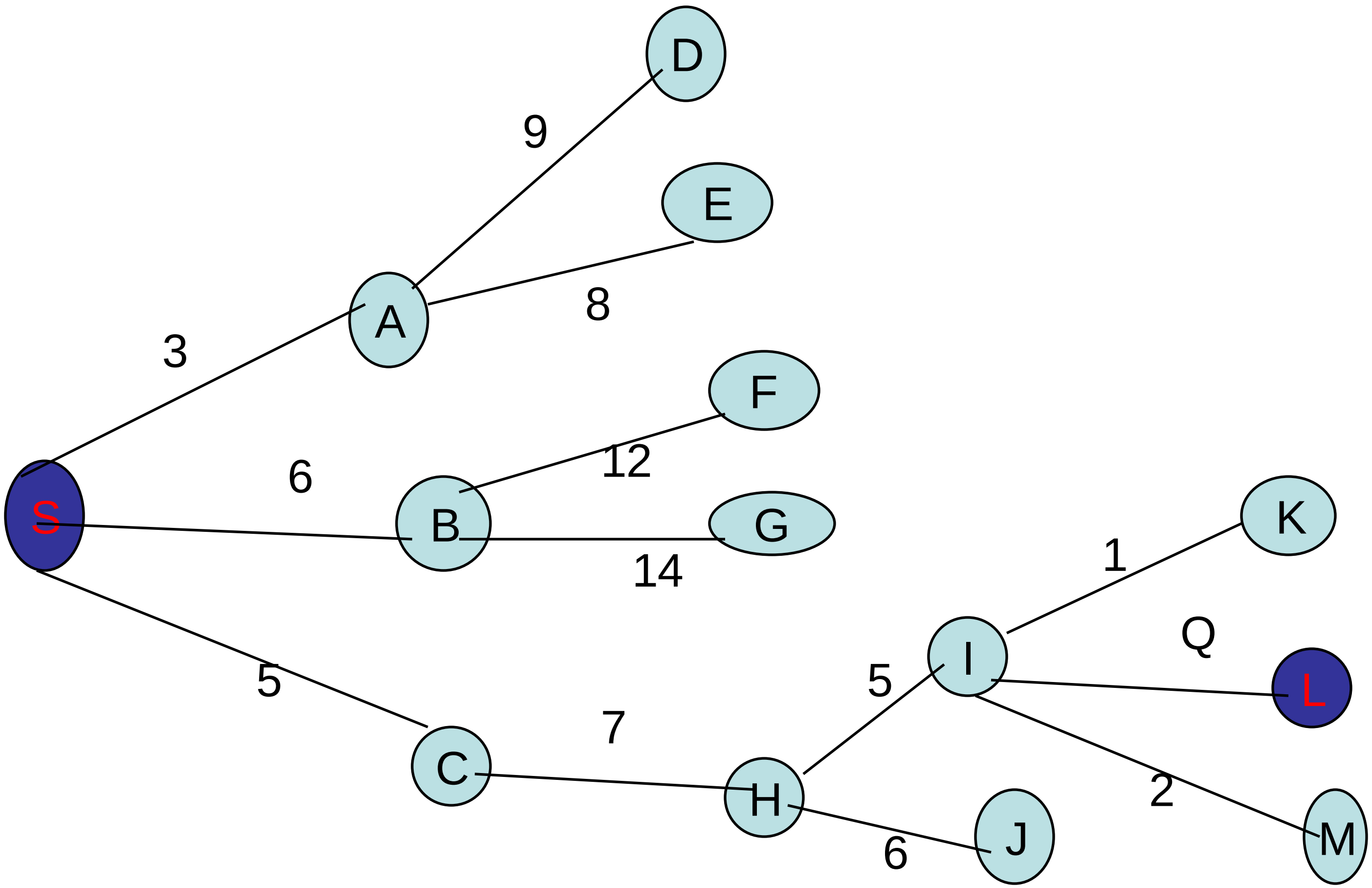
- One can prove: If *T* decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1
-
- Widely used in VLSI layout, airline scheduling, etc
-

Best first search

- A combination of DFS & BFS.
- DFS is good because a solution can be found without computing all nodes and BFS is good because it doesn't get trapped in dead ends.
- The best first search allows us to switch between paths going the benefit of both approaches.

How it works

- The algorithm maintains two list, one containing a list of candidate yet to explore -- OPEN
- One containing a list of visited node – CLOSED
- Since all unvisited successor nodes of every visited node are included in the OPEN list.
- It takes the advantage s of both DFS and BrFS.—faster.


S
A
B
C
D
E
F
G
H
I
J
K
L
M
3
6
5
9
8
12
14
7
5
6
1
Q
2

Step	Node being expanded	Children	Available Node	Node chose
1	S	(A:3)(B:6)(c:5)	(A:3)(B:6)(c:5)	(A;3)
2	A	(D:9)(E:8)	(B:6)(c:5) (D:9)(E:8)	(C:5)
3	C	(H:7)	(B:6) (D:9) (E:8) (H:7)	(B:6)
4	B	(E:12) (G:14)	(E:12) (G:14) (D:9) (E:8) (H:7)	(H:7)
5	H	(I;5) (J:6)	(E:12) (G:14) (D:9) (E:8) (I;5) (J:6)	(I:5)
6	I	K L M	All	L

A * algorithm

- This algorithm was given by hart Nilsson & Rafael in 1968.
- A* is a best first search algorithm with

f(n) = g(n) + h(n)

Where

g(n) = sum of edge costs from start to n

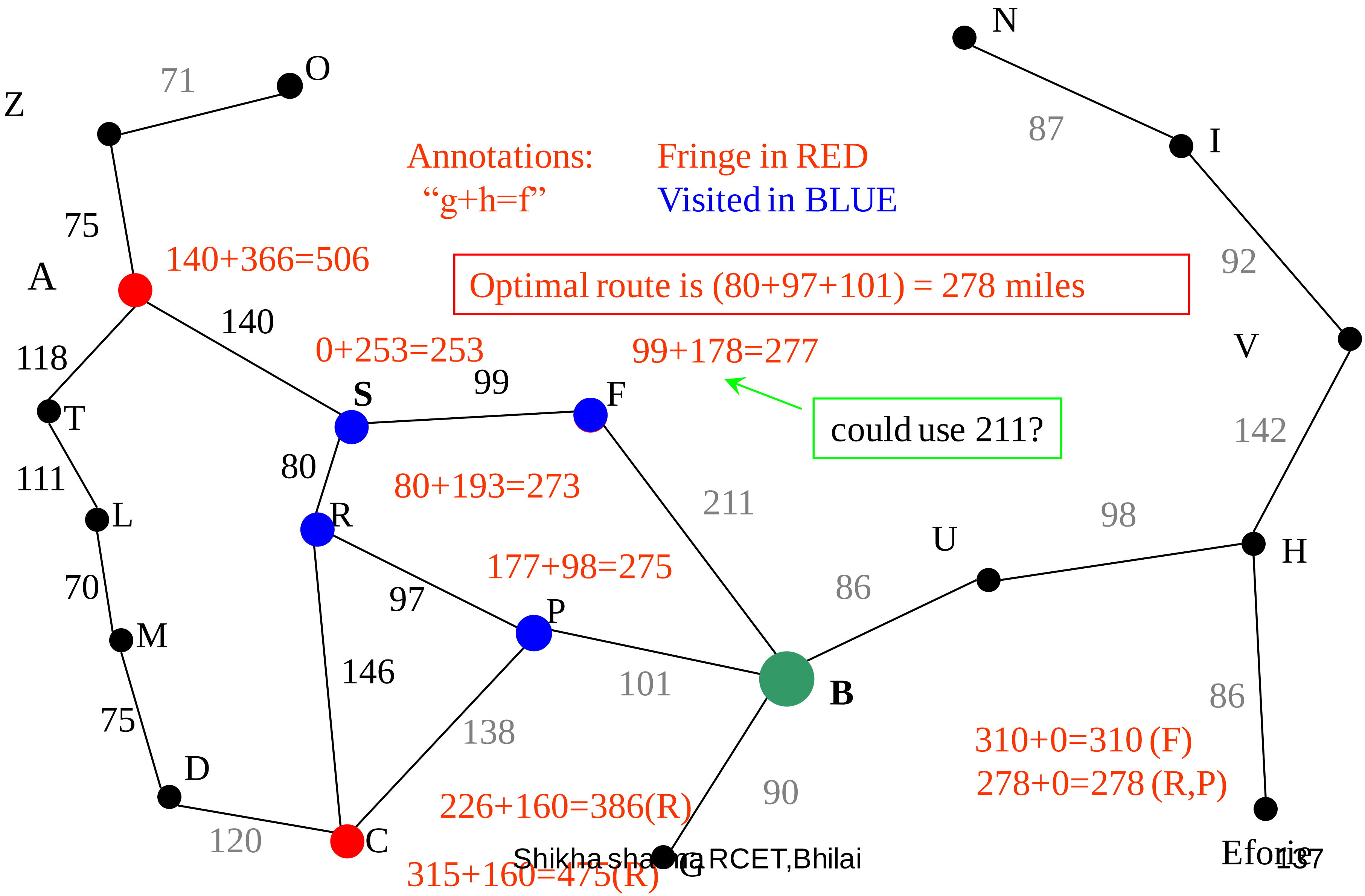
h(n) = estimate of lowest cost path from n to goal

- *f(n) = actual distanance so far + estimated distance remaining*

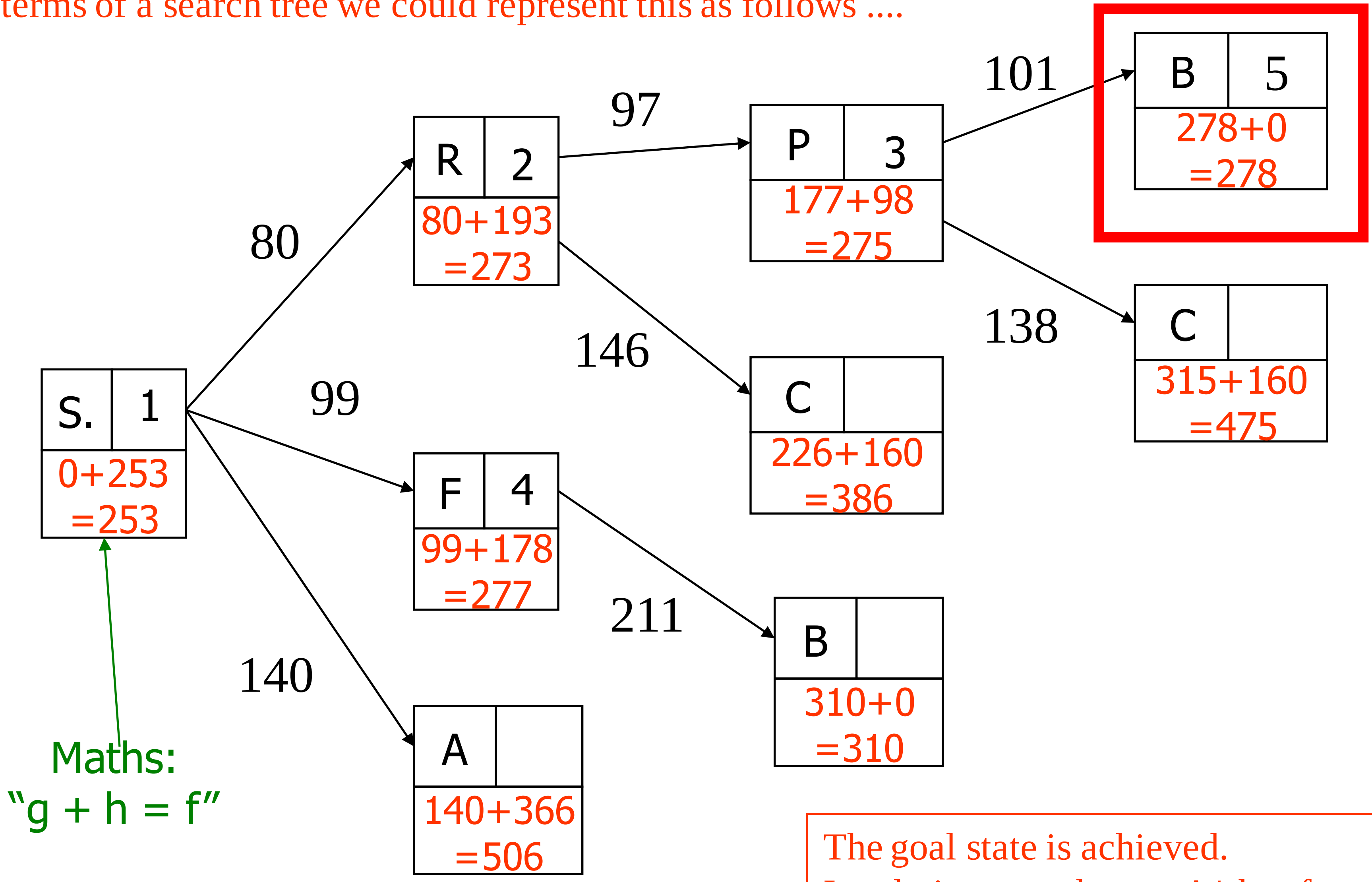
ANIMATION OF A*.

Nodes Expanded

1.S 2.R 3.P 4.F 5.B 278 GOAL!!



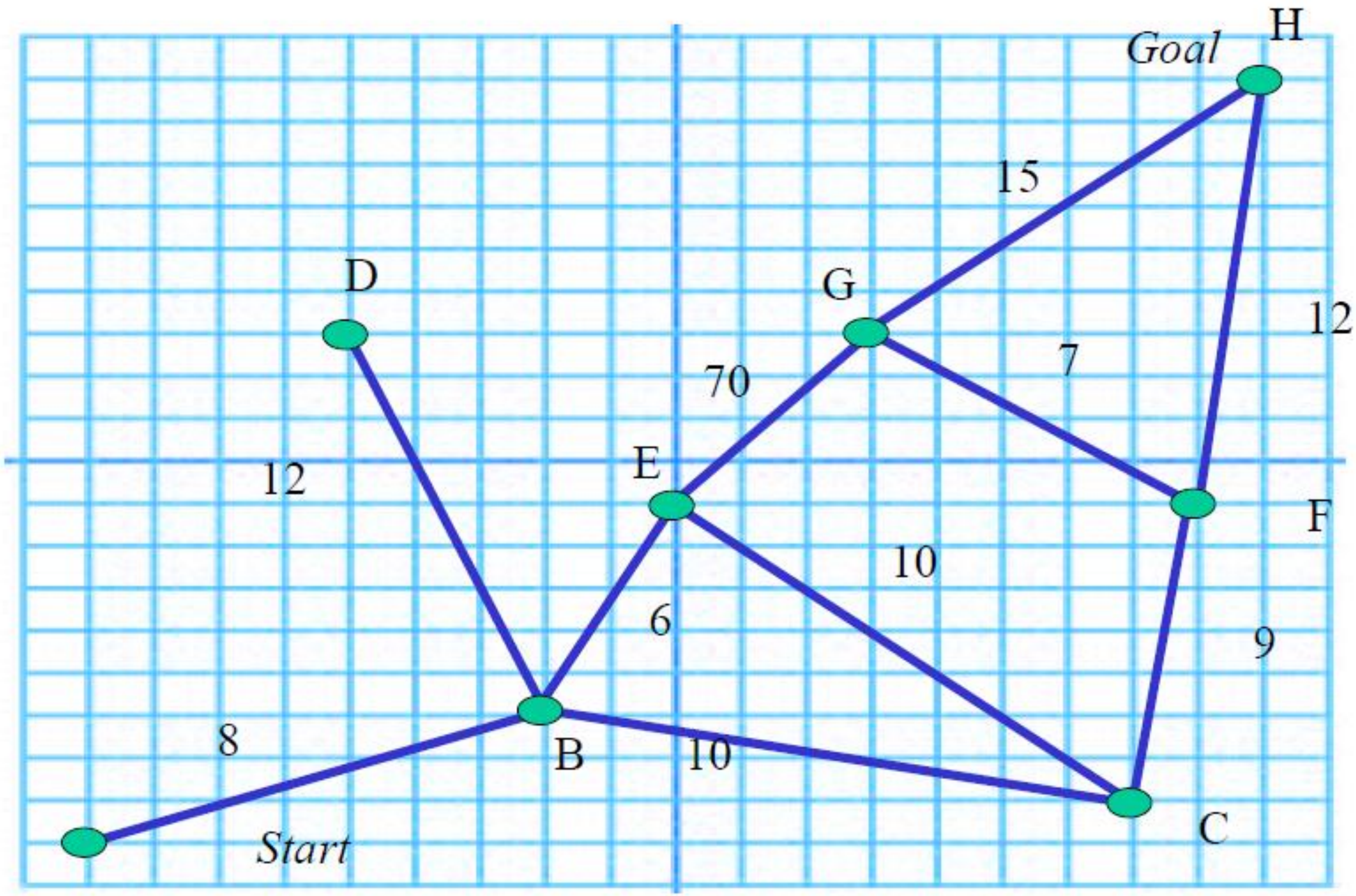
In terms of a search tree we could represent this as follows ....



The goal state is achieved.
In relation to path cost, A* has found the optimal route with 5 expansions.
Press space to end.

Press space to begin the search

A* SEARCH TREE


Goal
H
15
D
G
12
7
70
E
12
F
10
6
9
8
B
10
C
Start

Steps	Fringe	Node expanded	Comments
1	A		
2	B(26.6)	A	
3	E(27.5), C(35.1), D(35.2)	B	
4	C(35.1), D(35.2), ~~C(41.2)~~ G(92.5)	E	C is not inserted as there is another C with lower cost.
5	D(35.2), F(37), G(92.5)	C	
6	F(37), G(92.5)	D	
7	H(39), G(42.5)	F	G is replaced with a lower cost node
8	G(42.5)	H	Goal test successful.

Memory Usage of A*

- We store the tree in order to
  - to return the route
  - avoid repeated states
- Takes a lot of memory
- But scanning a tree is better with DFS

- **Means-Ends Analysis** (MEA)

- Means-Ends Analysis (MEA) is a technique used in AI for controlling search in problem solving computer programs.
- It is also a technique used at least since the 1950s as a creativity tool, most frequently mentioned in engineering books on design methods. Means-Ends Analysis is also a way to clarify one's thoughts when embarking on a mathematical proof.

Problem-solving as search

- An important aspect of intelligent behavior as studied in AI is *goal-based* problem solving, a framework in which the solution of a problem can be described by finding a sequence of *actions* that lead to a desirable goal. A goal-seeking system is supposed to be connected to its outside environment by or sensory, channels through which it receives information about the environment and motor, channels through which it acts on the environment. (The term "afferent" is used to describe "inward" sensory flows, and "efferent" is used to describe "outward" motor commands.) In addition, the system has some means of storing in a *memory* information about the *state* of the environment (afferent information) and information about actions (efferent information). Ability to attain goals depends on building up associations, simple or complex, between particular changes in states and particular actions that will bring these changes about. Search is the process of discovery and assembly of sequences of actions that will lead from a given state to a desired state. While this strategy may be appropriate for machine learning and problem solving, it is not always suggested for humans (e.g. cognitive load theory and its implications).

How MEA works

- The MEA technique is a strategy to control search in problem-solving. Given a current state and a goal state, an action is chosen which will reduce the *difference* between the two. The action is performed on the current state to produce a new state, and the process is recursively applied to this new state and the goal state.
- Note that, in order for MEA to be effective, the goal-seeking system must have a means of associating to any kind of detectable difference those actions that are relevant to reducing that difference. It must also have means for detecting the progress it is making (the changes in the differences between the actual and the desired state), as some attempted sequences of actions may fail and, hence, some alternate sequences may be tried.

- When knowledge is available concerning the importance of differences, the most important difference is selected first to further improve the average performance of MEA over other brute-force search strategies. However, even without the ordering of differences according to importance, MEA improves over other search heuristics (again in the average case) by focusing the problem solving on the actual differences between the current state and that of the goal.

- Means-Ends Analysis, a technique that was first used in Newell and Simon's General Problem Solver (GPS), is a problem-solving technique in which the current state is compared to the goal state, and the difference between them is divided up into sub goals in order to achieve the goal state by the use of the available operators.
- Means-End analysis is one of many weak search methods that have been utilized in both cognitive architectures and more general artificial intelligence research.

Constraint Satisfaction

What is a constraint problem?

- A constraint problem is a task where you have to
  - Arrange objects
  - Schedule tasks
  - Assign values
  - …
  - subject to a number of constraints

Example of constraint problems

Cryptarithmetic problems:

```
   S E N D
 + M O R E
 _________
 M O N E Y
```

Each letter stands for a different digit. Assign digits to the letters so that the sum is correct.

Constraint: when the values are assigned, the sum must add up correctly.

Some easy examples

- AS + A = MOM
- I + DID = TOO
- A + FAT = ASS
- SO + SO = TOO
- US + AS = ALL
- ED + DI = DID
- DI + IS = ILL

1. CROSS

+ ROADS

DANGER

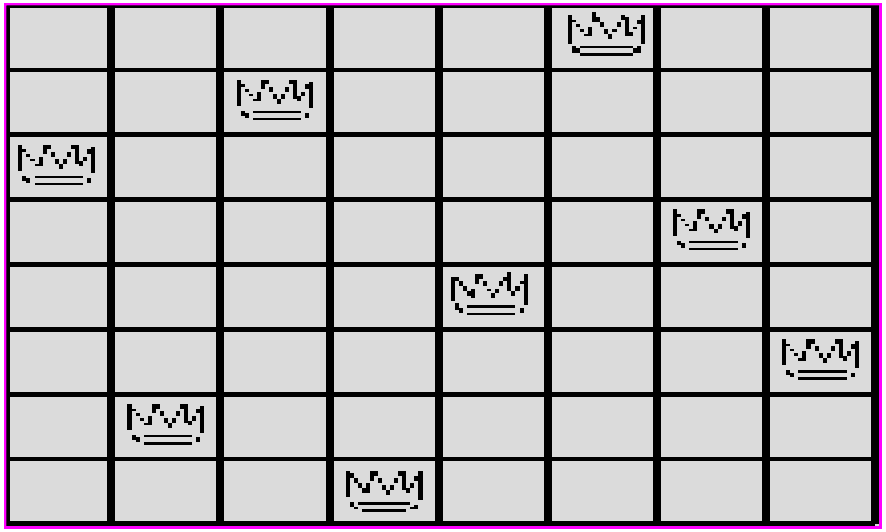
2.

TWO

+ TWO

FOUR

Another example

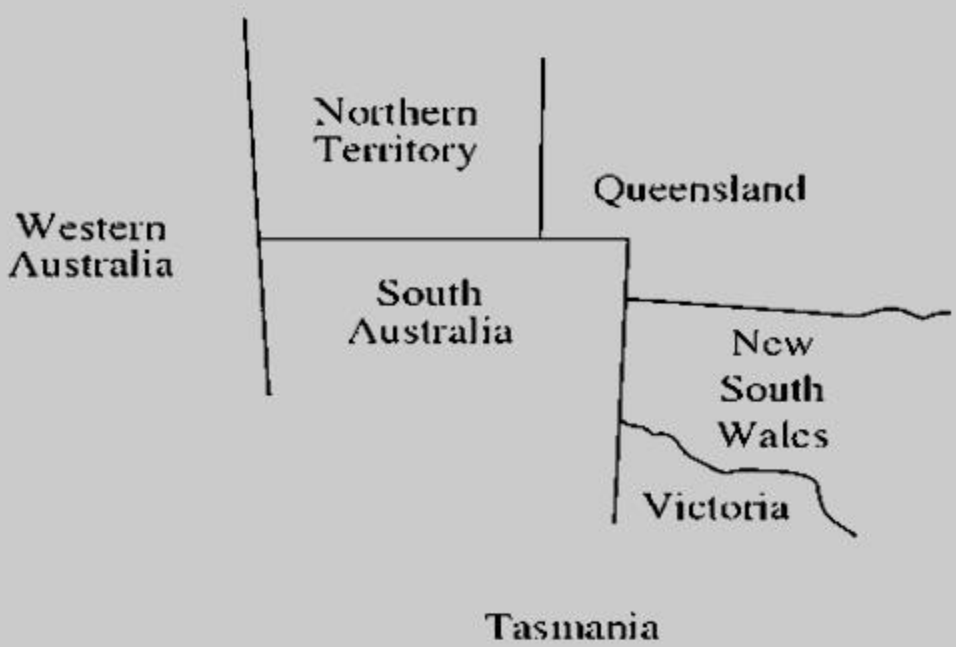


The 8 Queens puzzle

Place 8 queens on a chessboard so that no two queens are attacking one another.

Constraints: no two queens must be on the same row, the same column, or the same diagonal

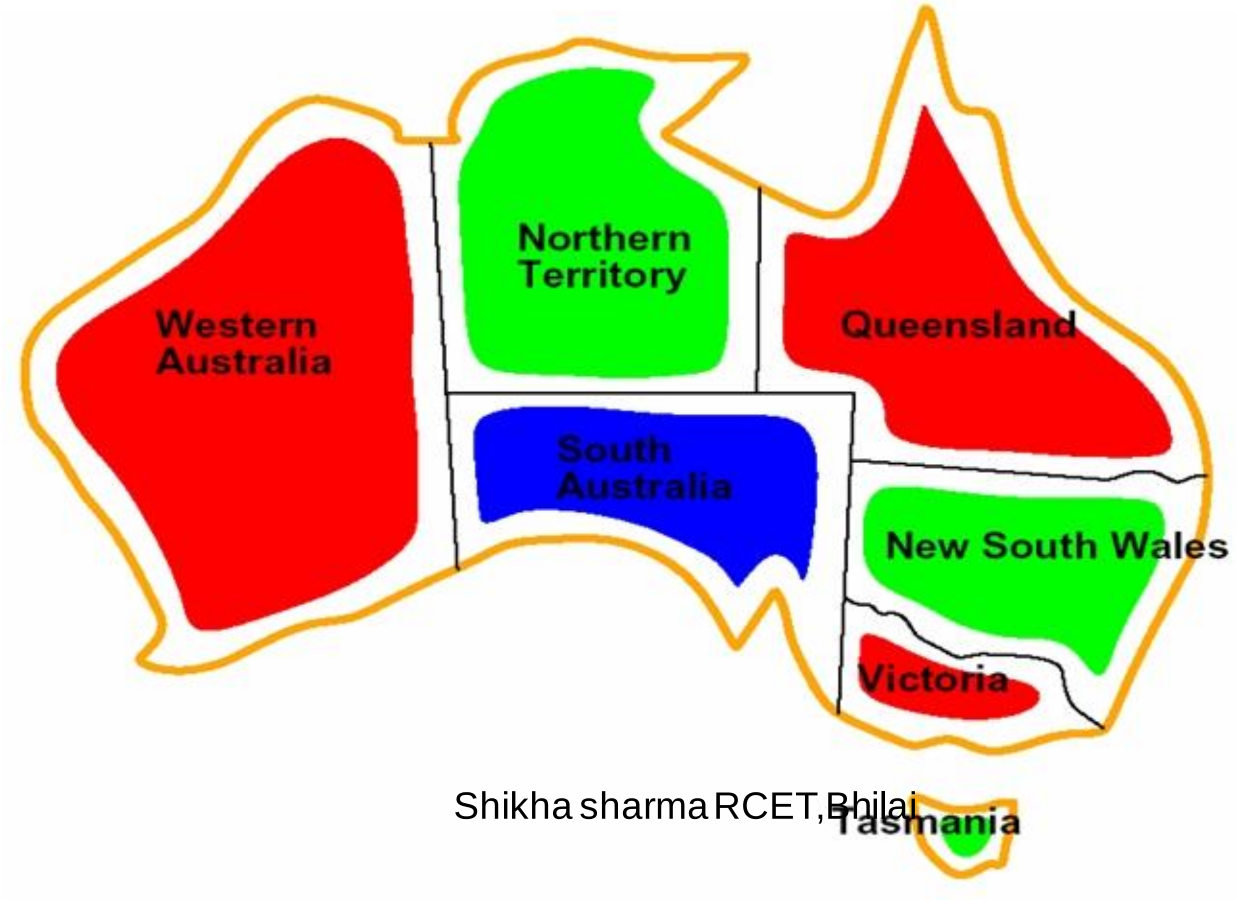
Example: Map-Coloring Problem



- Variables: WA, NT, Q, NSW, V, SA, T
- Domains: D_i= {red, green, blue}
- Constraints: neighboring regions must have different colors

Example: Map-Coloring Problem

- Solutions: assignments satisfying all constraints, e.g.,
  {WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green}



A more practical example

- Timetabling/scheduling
  - Assign classes to rooms so that
    - Students aren't required to be in two different rooms at the same time
    - Similarly for lecturers
    - Two classes aren't booked into the same room at the same time
    - Rooms are sufficiently large to hold classes assigned to them
    - Labs have enough computers for the classes assigned to them
    - …

Formal definition of a constraint problem

- A constraint problem consists of
  - A set of variables x_1, x_2,… x_n
  - For each variable x_i a finite set D_i of its possible values (its domain)
  - A set of constraints restricting the values that the variables can take
- Goal: find an assignment of values to the variables which satisfies all the constraints

Summary

- Constraint problem-solving can be applied to a wide variety of real-world problems
- Formally, a constraint problem consists of
  – A set of variables and their domains
  – A set of constraints
- The goal
  – Find a valid set of values
  – Find all sets of values
  – Find the best set of values
- The method
  – Combine search and constraint propagation

SOLUTIONS

```
    SEND
 +  MORE
__________
MONEY
```

```
      9 5 6 7
 +    1 0 8 5
    1 0 6 5 2
```

U S
\+ A S
A L L

8 5
\+ 1 5
1 0 0

Game Playing Algorithms

Road Map

- Motivation behind using AI techniques for Game Playing.
- Categories Of Search
- Search Strategies
- Minmax Search Procedure
- Branch & Bound Technique for limiting search space (alpha-beta cutoff)
- Iterative Deepening
- Othello

Games and AI

- Games were one of the first tasks undertaken by researchers in AI field
- A. Turing wrote chess playing program in 1950's
- Why research on games continues?
  - ➢ Long-standing fascination for games
  - ➢ Some difficult games remain to be won by computers

Motivation

- Some games provide challenges that can be formulated as abstract competitions with clearly defined states and rules.
- Games can be used to demonstrate the power of computer-based techniques and methods.
- More challenging games require the incorporation of specific knowledge and information.

Who is better?

- Machines are better than humans in: Othello
- Machines and humans are about equally good in: Backgammon, Scrabble
- Humans are better than machines in: Go, Bridge

Game Trees

- Formal Description of Game :
  - Initial State
  - Successor function
  - Terminal State
  - Utility function
- Games are represented by game trees in which
  - Each node represents a position
  - Each link represents a legal move
  - Leaf nodes are final positions(Win,Loss or Draw)
  - The aim is to reach the goal node from the root node.

Types of Games

- Two player vs. Multiplayer
  Tic-Tac-Toe vs. Bridge
- Zero-sum vs. General-sum
  Chekers vs. Auction
- Perfect information vs. Imperfect information
  Othello vs. Bridge
- Deterministic vs. Chance
  Chess vs. Backgammon

Search Procedures

- Generate using simple legal-move generator will result in very large testing space for the tester.
- So use ***plausible*** move generator.
- Now test procedure can spend more time evaluating each of the moves, so more reliable results.

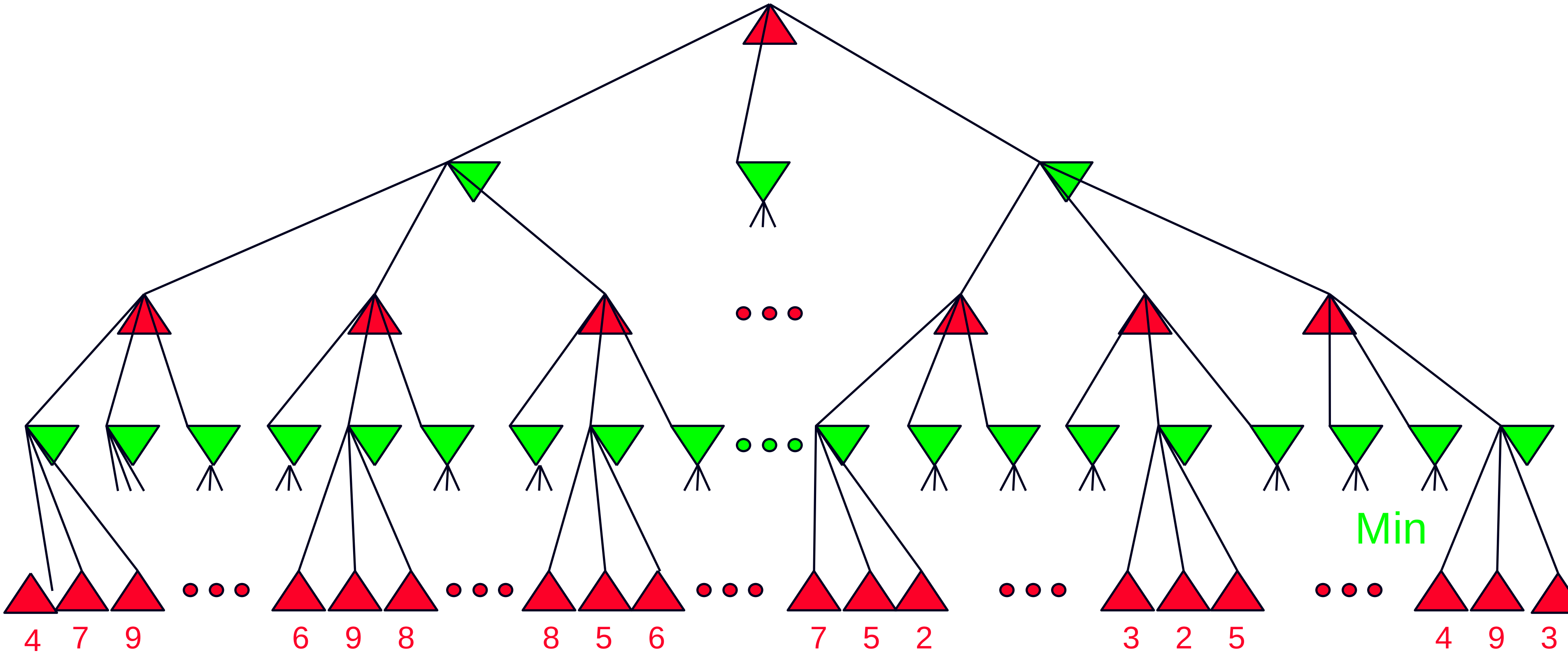
Search Procedures

- In order to choose the best move, the resulting board position must be compared to discover which is most advantageous -
  - Use ***Static Evaluation Function (Utility Function)***
- ***It estimates how likely the particular state can eventually lead to a win.***

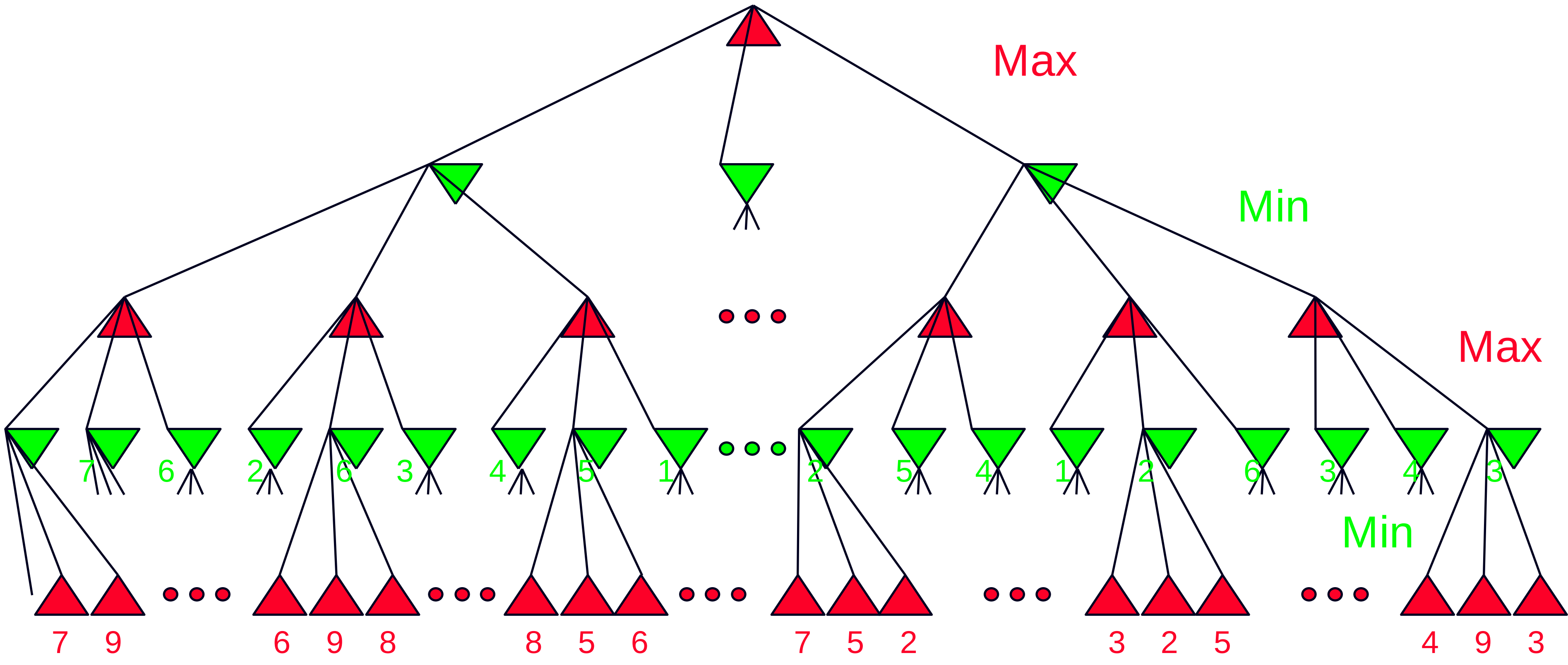
Minimax Search Procedure

- Depth-limited.
- Use plausible move generator to generate set of possible successor positions.
- Apply static evaluation function to those positions & choose the best one.
- Back up that value to the starting point.

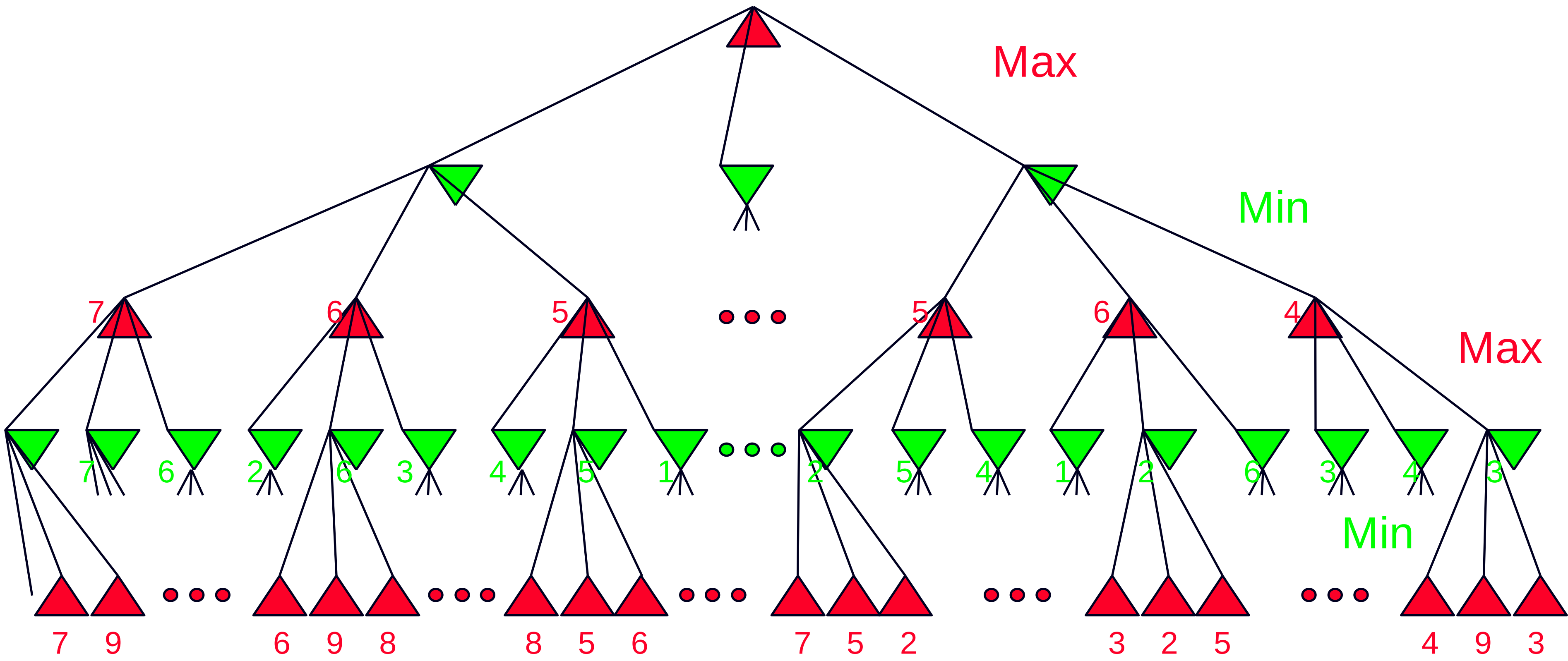
Minimax Example



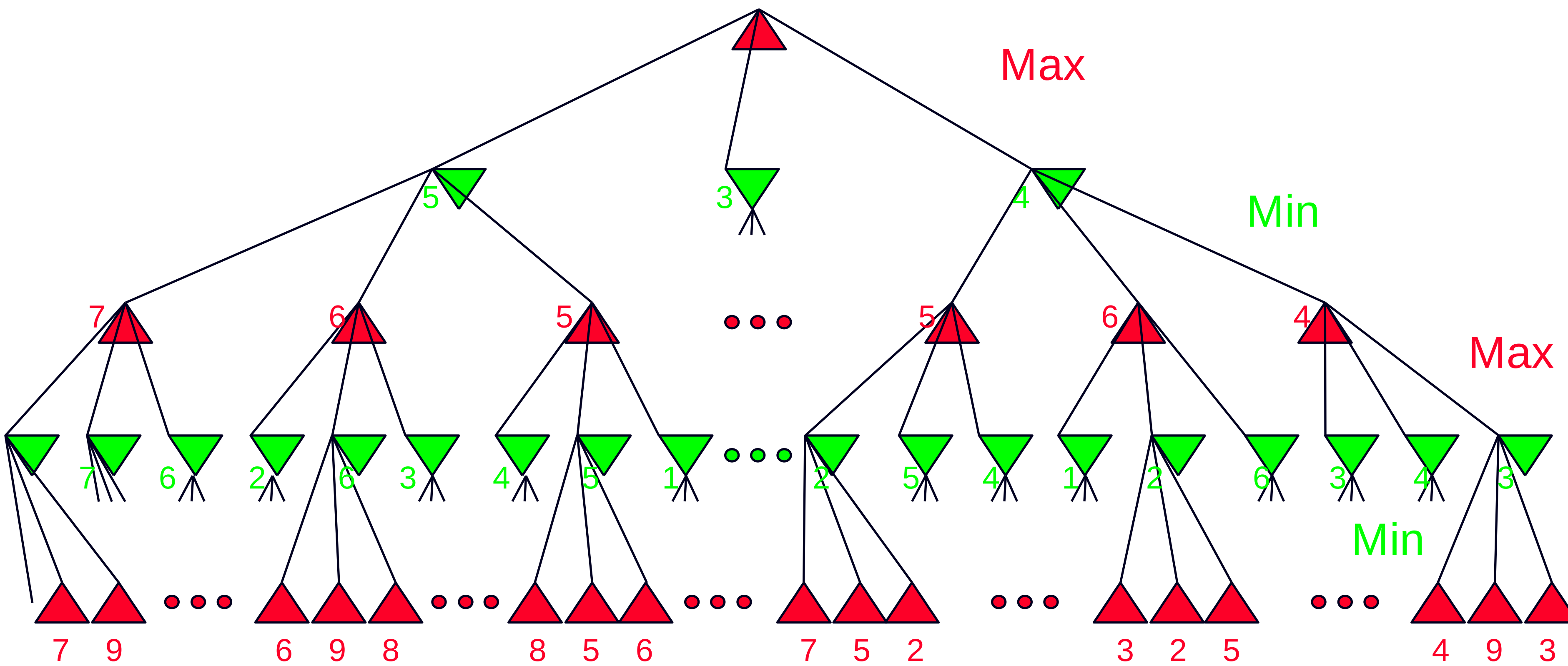
Minimax Example



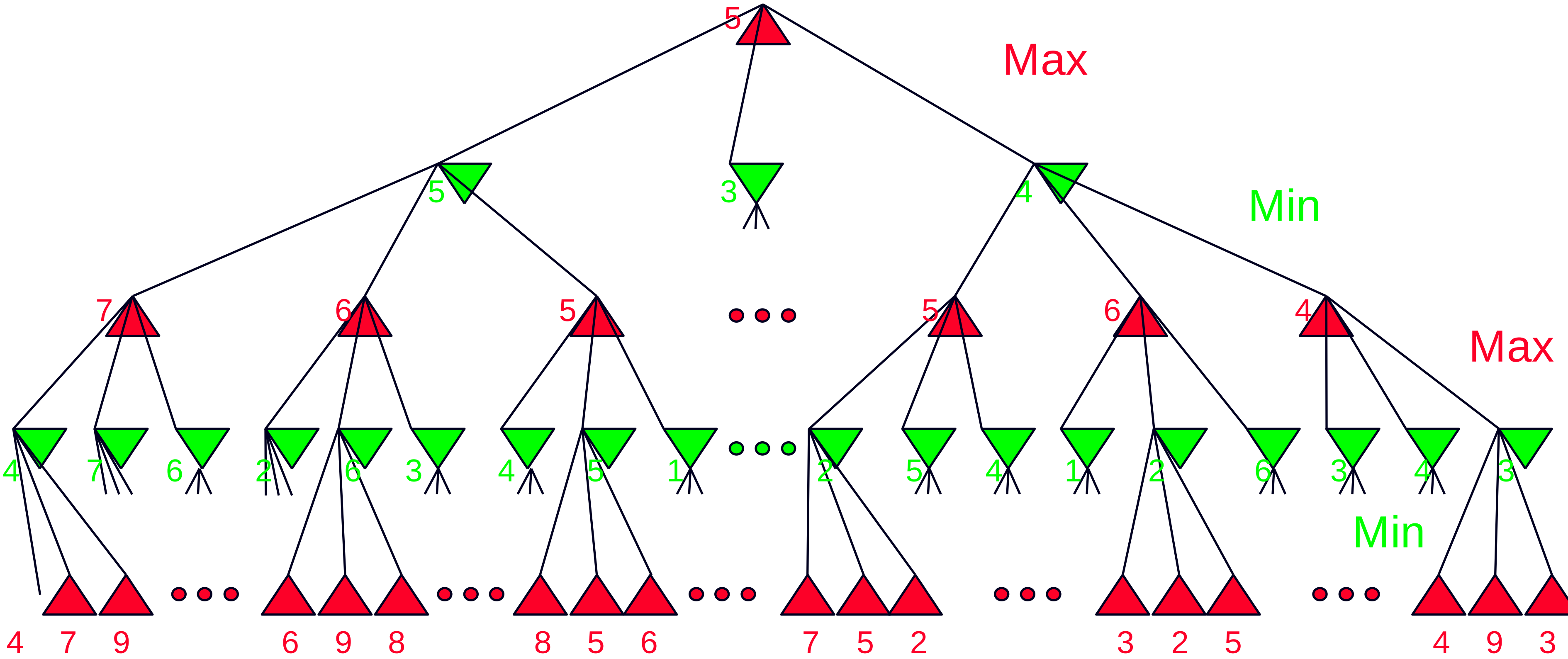
Minimax Example



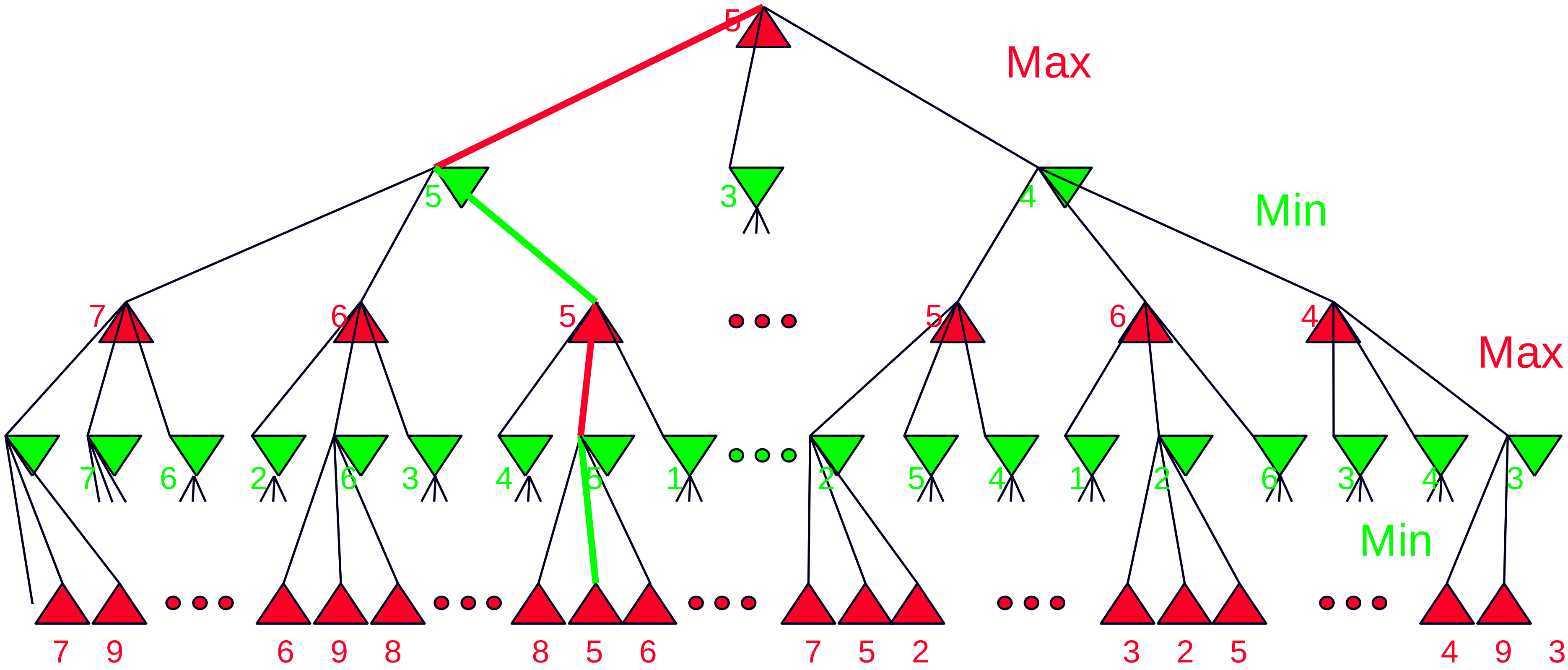
Minimax Example



Minimax Example



Minimax Example



Adding Alpha-Beta Cutoffs

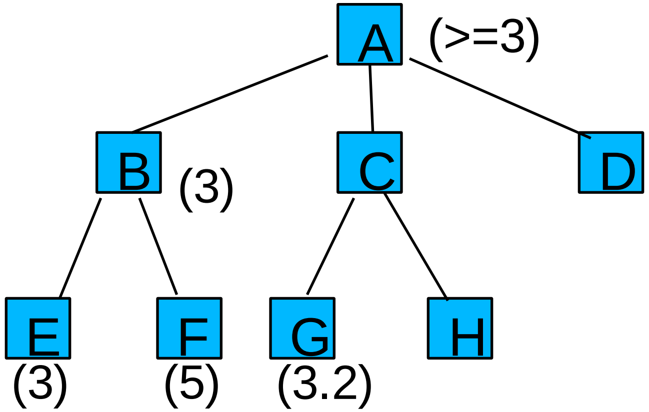
- Minimax is a depth-first process, its efficiency can be improved by using *Branch & Bound Technique.*
- *Partial solutions that are clearly worse than known solutions can be abandoned.*
- *Requires maintanenence of 2 threshold values -*
  - *Alpha – lower bound on the value that a maximized node may be assigned.*
  - *Beta – upper bound on the value that a minimizing node may be assigned.*

Adding Alpha-Beta Cutoffs

- Search at the minimizing level can be terminated when a value less than alpha is discovered.
- Search at the maximizing level can be terminated when a value greater than beta is discovered.
- At maximizing levels, only beta is used to determine whether to cut-off the search & similarly for minimizing levels.

Futility Cutoff

- Cutoff additional paths that appear to be at best only slight improvements over paths that have already been explored.



The best we can hope after examining node G is move C with a score of 3.2. Move B guarantees a score of 3. Since 3.2 is only slightly better than 3, we should terminate exploration of C.

Additional Refinements for Mini-max

- Waiting for Quiescence
  - Continue the search until no drastic change occurs from one level to next.
  - Using Book Moves
  - Use book moves in opening sequences & endgame sequences, combined with minimax procedure for the midgame.

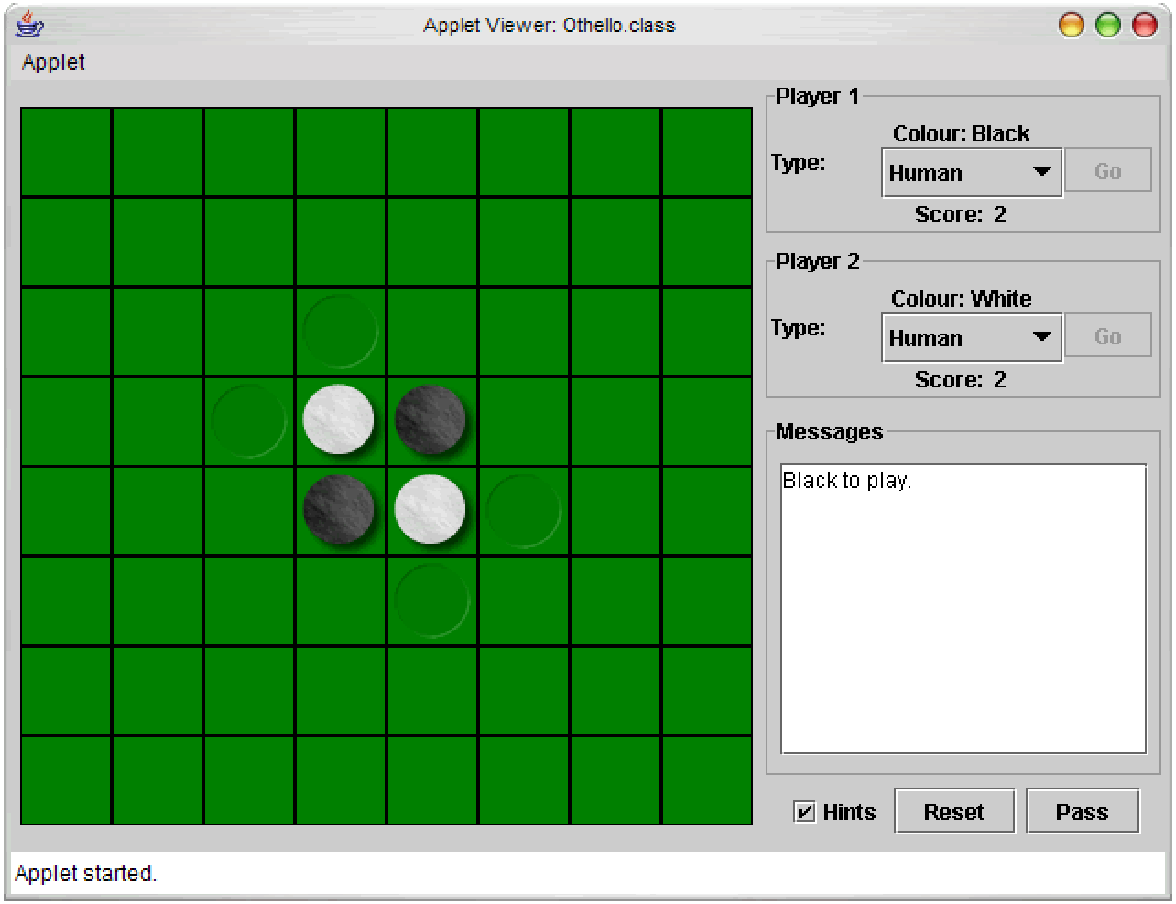
Iterative Deepening

- Neither suffers with the drawbacks of breadth-first nor depth-first search
- Perform a depth-first search to depth one
- Discarding the nodes generated in the first search, start over and do a depth-first search to level two
- Next, start over again and do a depth-first search to depth three, etc., continuing this process until a goal state is reached.
- Guaranteed to find a shortest-length solution

Iterative Deepening

- At any given time it is performing a depth-first search, and never searches deeper than depth d, so the space it uses is O(d).
- Disadvantage is that it performs wasted computation prior to reaching the goal depth.
- This wasted computation does not affect the asymptotic growth of the run time for exponential tree search
- Intuitive reason is that almost all the work is done at the deepest level of the search

Othello



Applet Viewer: Othello.class
Applet
Player 1
Colour: Black
Type:
Human
Go
Score: 2
Player 2
Colour: White
Type:
Human
Go
Score: 2
Messages
Black to play.
Hints
Reset
Pass
Applet started.

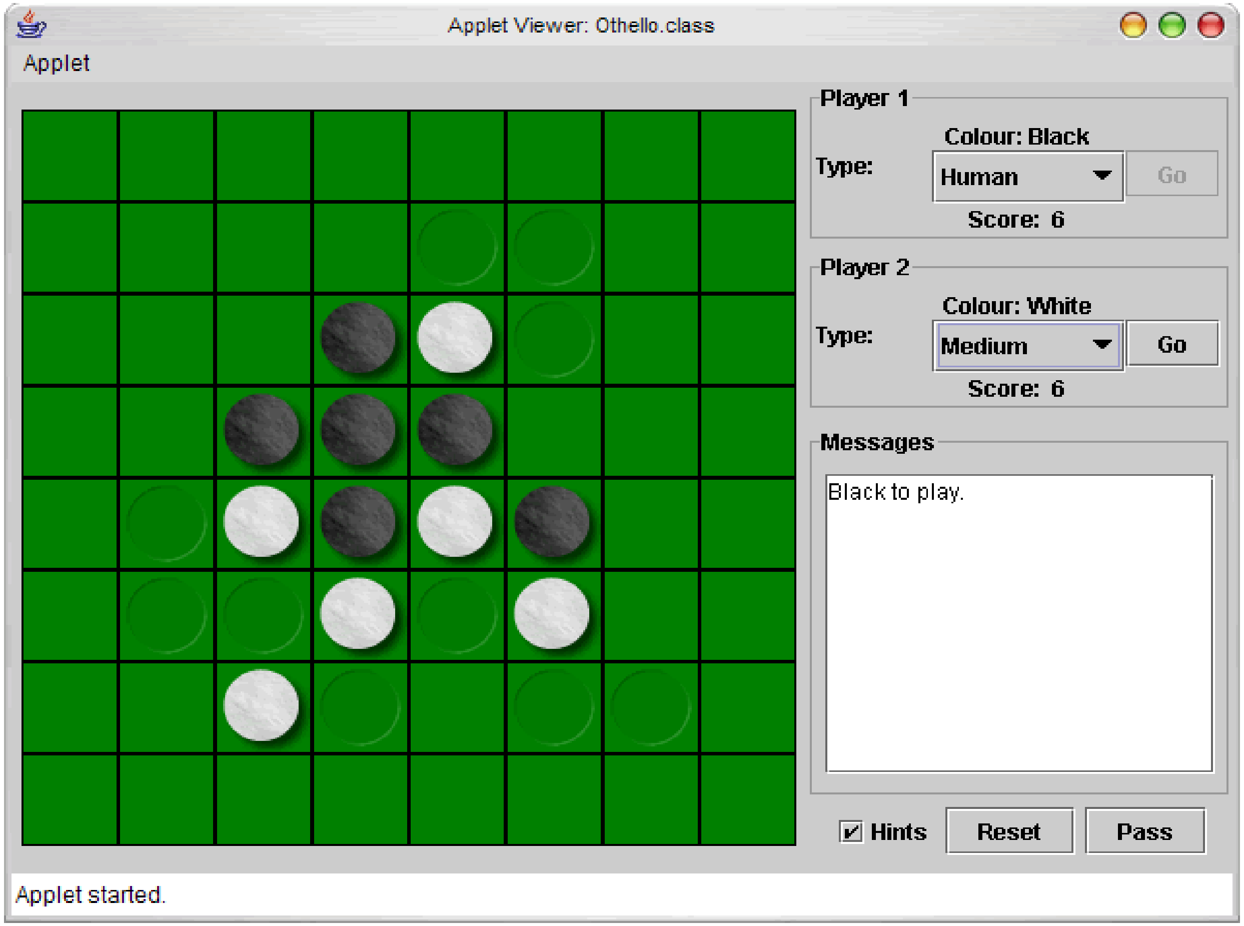
Othello Rules

- Two Players - Black and White
- Initial Position
  - White owns the two central squares on the main diagonal
  - Black owns the two central squares on the minor diagonal
- Rules
  - Black plays first, and then the players take turns moving until neither side has a legal move

Othello Rules

- At this point, the player with the most discs of his color is declared the winner (there may be ties).
- Legal Move
  - Player takes a turn by placing a disc of his color in an empty position.
  - He must ensure that one or more of the opponent's discs are sandwiched between the newly placed disc and his another disc.
  - The opponent's discs that are surrounded then change to his colour.

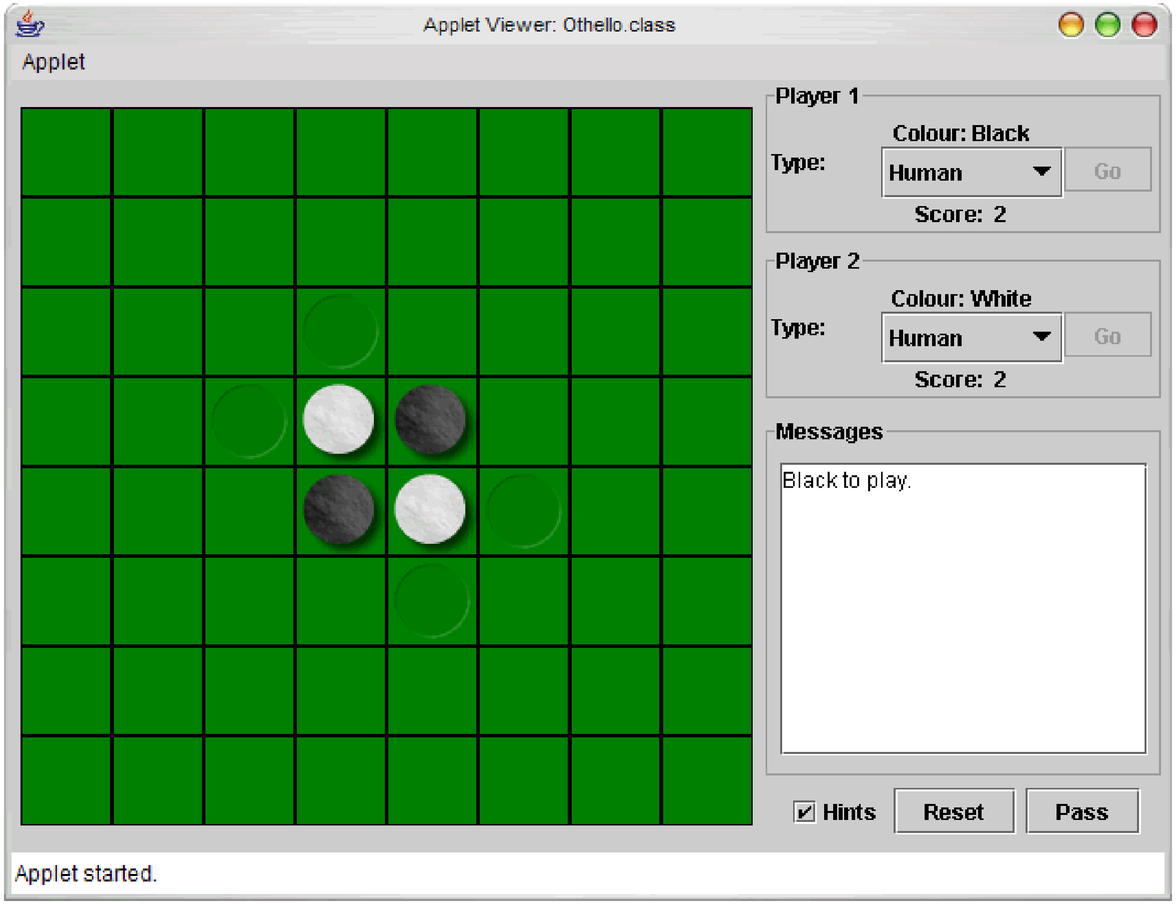
Othello Example



Strategies in Othello

- Maximizing the discs
  - Play the move which captures the most discs.
- Weighted square strategy
  - The weighted square strategy stems from the observation that not all of the squares on the Othello board are of equal value.

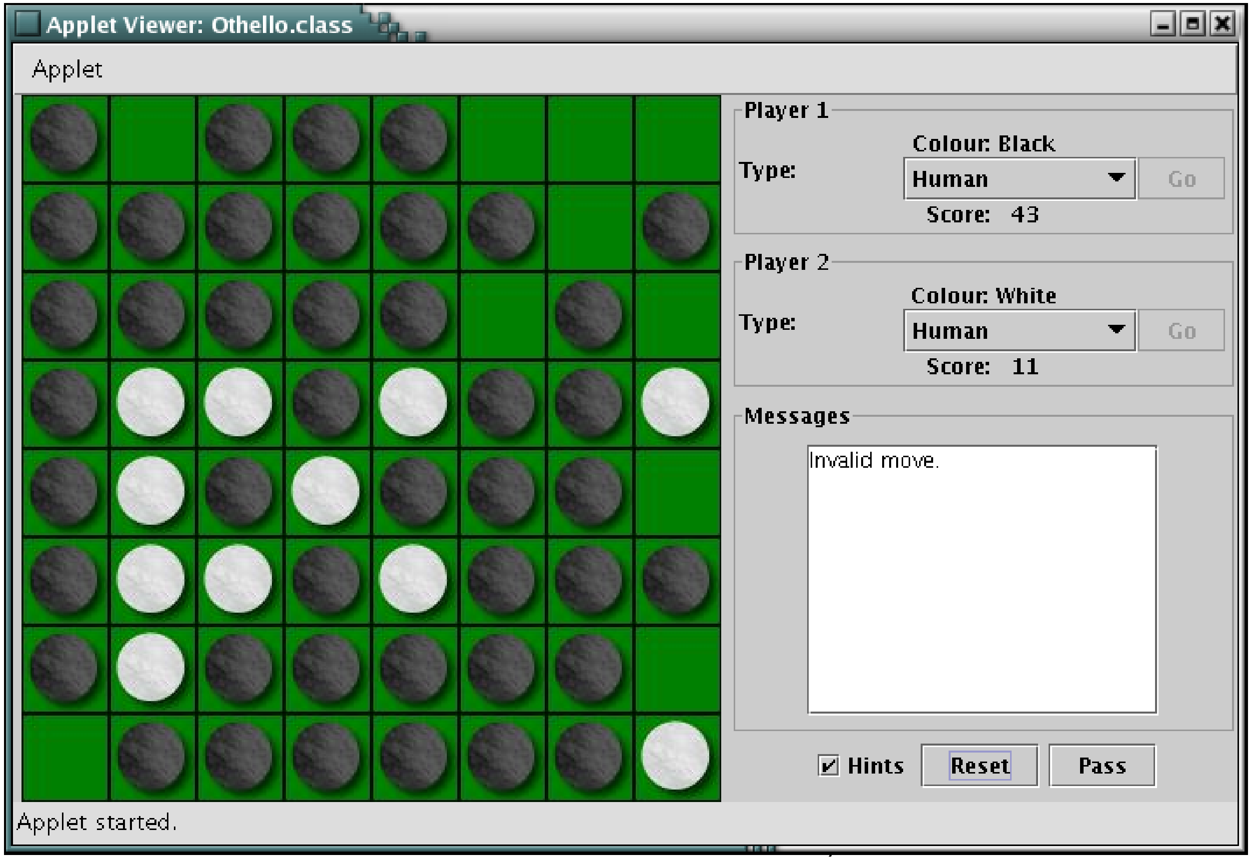
Strategies in Othello



Strategies in Othello

- Maximising the discs
- Weighted square strategy
- Mobility

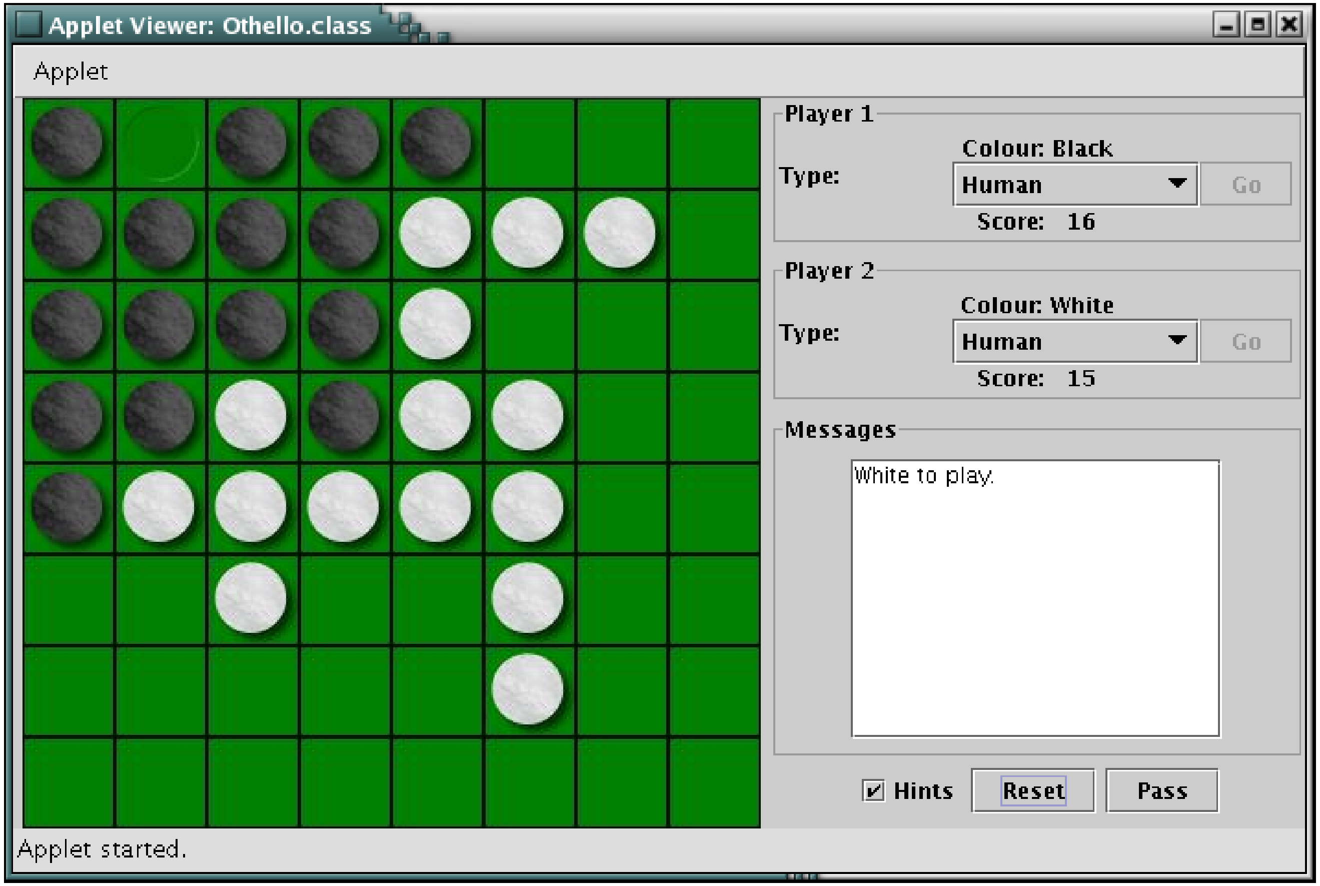
Strategies in Othello: Mobility



Strategies in Othello

- Maximising the discs
- Weighted square strategy
- Mobility
- Stable discs
  - A completely stable disc can never be recaptured (flipped) by the opponent.

Strategies in Othello : Stability



Othello Implementations

- IAGO:Rosenbloom,1982
  - Book Moves
- BILL:Lee and Mahajan,1990
  - Table based evaluation
  - Iterative deepening
- LOGISTELLO:Michael Buro,1997
  - PROBCUT
  - Automated machine learning

Conclusion

- The unending fasination towards games is the important motivation behind the immense progress in the field of game playing.
- Machine games provide us new challenges and also act as an excellent tutor. Hence, this area is flourishing along with all other areas of AI with equal pace.
- It also led us in developing high performance, real time solutions to the problems in various other fields like robotics.