Unit: 2
AI Problems and Search

Outline

- Problems,
- Problem Spaces and Search: Problem as state space search,
- Production systems
- Problem Characteristics
- Heuristic Search Techniques:
  - Hill Climbing,
  - Best First Search and A*
  - Problem Reduction and AO*
  - Constraint Satisfaction
  - Means-Ends Analysis

Introduction

Problem and Problem Solving

- The steps that are required to build a system to solve a particular problem are:
  1. Problem Definition that must include precise specifications of what the initial situation will be, as well as what final situations constitute acceptable solutions to the problem.
  2. Problem Analysis, this can have immense impact on the appropriateness of various possible techniques for solving the problem.
  3. Isolate and Represent the task knowledge required to solve the problem.
  4. Selection of the best technique(s) for solving the particular problem.

- Problem solving is a process of generating solutions from the observed data.

State and State Space Representation

- A state is a representation of problem elements at a given moment.
- A state space is the set of all possible states reachable from the initial state.
- A state space forms a graph in which the nodes are states and the arcs between nodes are actions.
- In a state space, a path is a sequence of states connected by a sequence of actions.
- The solution of a problem is a part of the graph formed by the state space.

Define the Problem as State Space Search

- To provide a formal description of a problem, we need to do the following:
  1. Define a state space that contains all the possible configurations of the relevant objects.
  2. Specify one or more states that describe possible situations, from which the problem solving process may start. These states are called initial states.
  3. Specify one or more states that would be acceptable solution to the problem. These states are called goal states.
- Specify a set of rules that describe the actions (operators) available.
- The problem can then be solved by using the rules, in combination with an appropriate control strategy, to move through the problem space until a path from an initial state to a goal state is found. This process is known as 'search'.

State Space Representation – Water Jug

- Problem Definition: You are given two jugs, a 4-gallon one and a 3-gallon one, a pump which has unlimited water which you can use to fill the jug, and the ground on which water may be poured. Neither jug has any measuring markings on it. How can you get exactly 2 gallons of water in the 4-gallon jug?

1. Initial State
   - We will represent a state of the problem as a tuple (x, y), where x represents the amount of water in the 4-gallon jug and y represents the amount of water in the 3-gallon jug.
   - Note that $0 \leq x \leq 4$, and $0 \leq y \leq 3$.
   - Here the initial state is (0, 0). The goal state is (2, n) for any value of n.

State Space Representation – Water Jug

2. Production Rules

Sr.	Current state	Next state	Description
1	(x, y) If x<4	(4, y)	fill the 4- gallon jug
2	(x, y) If x<3	(x,3)	fill the 3-gallon jug
3	(x, y) If x>0	(x-d, y)	pour some water out of the 4- gallon jug
4	(x, y) If y>0	(x, y-d)	pour some water out of the 3- gallon jug
5	(x, y) If x>0	(0, y)	empty the 4- gallon jug on the ground
6	(x, y) If y>0	(x, 0)	empty the 3- gallon jug on the ground

State Space Representation – Water Jug

2. Production Rules

Sr.	Current state	Next state	Description
7	(x, y) If x + y >= 4 & y>0	(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 >= 3 & x>0	(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 <= 4 & y>0	(x+y,0)	pour all the water from the 3-gallon jug into the 4-gallon jug

State Space Representation – Water Jug

2. Production Rules

Sr.	Current state	Next state	Description
10	(x, y) If x + y <= 3 & x>0	→ (0,x+y)	pour all the water from the 4 -gallon jug into the 3-gallon jug
11	(0,2)	→ (2,0)	pour the 2-gallon from the 3 – gallon jug into the 4-gallon jug
12	(2,y)	→ (0, y)	empty the 2 gallon in the 4 gallon on the ground

Water Jug – Solution

3. Productions for the water jug problem

Gallons in the 4- gallon Jug	Gallons in the 3- gallon	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

State Space Representation – 8 Puzzle

- Problem Definition: The 8 puzzle consists of eight numbered, movable tiles set in a 3x3 frame. One cell of the frame is always empty thus making it possible to move an adjacent numbered tile into the empty cell. Such a puzzle is illustrated in following diagram.

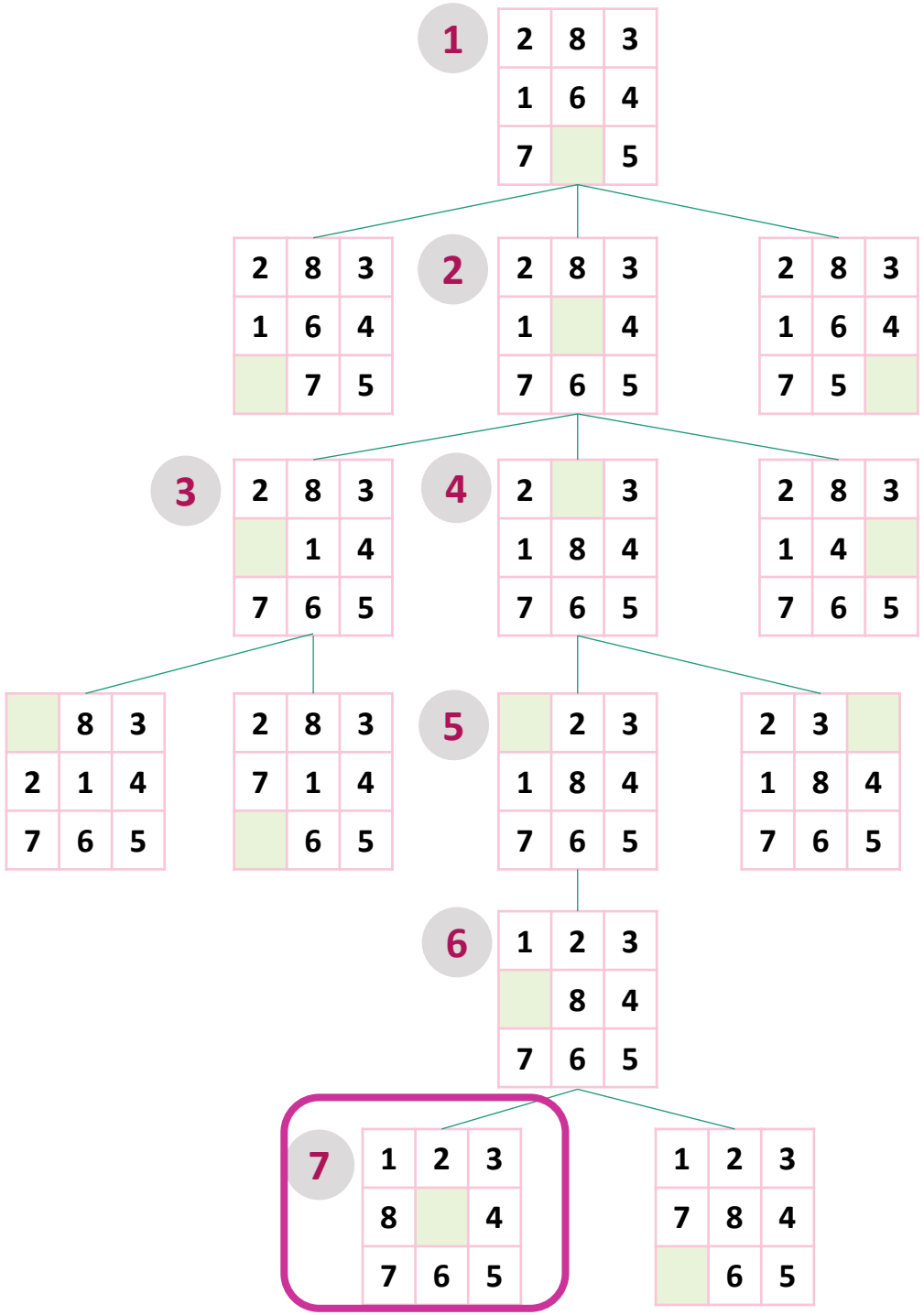
Initial State

Goal State

State Space Representation – 8 Puzzle

- A solution to the problem is an appropriate sequence of moves, such as “move tiles 5 to the right, move tile 7 to the left ,move tile 6 to the down” etc…

8 Puzzle - Game Tree



Problem Characteristics

1. Is the problem decomposable into a set of independent smaller or easier sub-problems?
2. Can solution steps be ignored or at least undone if they prove unwise?
3. Is the problem's universe predictable?
4. Is a good solution to the problem obvious without comparison to all other possible solutions?
5. Is the desired solution a state of the world or a path to a state?
6. Is a large amount of knowledge absolutely required to solve the problem or is knowledge important only to constrain the search?
7. Can a computer that is simply given the problem return the solution or will the solution of the problem require interaction between the computer and a person?

Chess Analysis with Respect to Seven Problem Characteristics

Problem Characteristics	Satisfied	Justification
Is the problem decomposable ?	No	Dependent moves
Can solution steps be ignored or at least undone	No	Wrong move can’t be undone
Is the problem’s universe predictable	No	Moves of other player can not be predicted
Is a good solution absolute or relative?	Absolute	Winning position need not be compared
Is the solution a state or a path?	Path	Not only solution but how it is achieved also matters
What is the role of knowledge?		Domain specific knowledge is required to constrain search
Does the task require Interaction with a person?	No	Once all rules are defined, no need for interaction

8 Puzzle Analysis with Respect to Seven Problem Characteristics

Problem Characteristics	Satisfied	Justification
Is the problem decomposable ?	No	Dependent moves
Can solution steps be ignored or at least undone	Yes	We can undo the previous move
Is the problem's universe predictable	Yes	Problem Universe is predictable, it is a single person game
Is a good solution absolute or relative?	Absolute	Winning position need not be compared
Is the solution a state or a path?	Path	Not only solution but how it is achieved also matters
What is the role of knowledge?		Domain specific knowledge is required to constrain search
Does the task require Interaction with a person?	No	In 8 puzzle additional assistance is not required

Production System

- Production systems provide appropriate structures for performing and describing search processes.
- A production system has four basic components:
  1. A set of rules each consisting of a left side that determines the applicability of the rule and a right side that describes the operation to be performed if the rule is applied.
  2. A database of current facts established during the process of inference.
  3. A control strategy that specifies the order in which the rules will be compared with facts in the database and also specifies how to resolve conflicts in selection of several rules or selection of more facts.
  4. A rule applier.
- Production systems provide us with good ways of describing the operations that can be performed in a search for a solution to a problem.

Production System Characteristics

1. A monotonic production system is a production system in which the application of a rule never prevents the later application of another rule that could also have been applied at the time the first rule was selected.
2. A non-monotonic production system is one in which this is not true. This production system increases the problem-solving efficiency of the machine by **not keeping** a record of the changes made in the previous search process.
3. A partially communicative production system is a production system with the property that if the application of a particular sequence of rules transforms state P into state Q, then any combination of those rules that is allowable also transforms state P into state Q.
4. A commutative production system is a production system that is both monotonic and partially commutative. These type of production systems is used when the order of operation is not important, and the changes are reversible.

Search Techniques

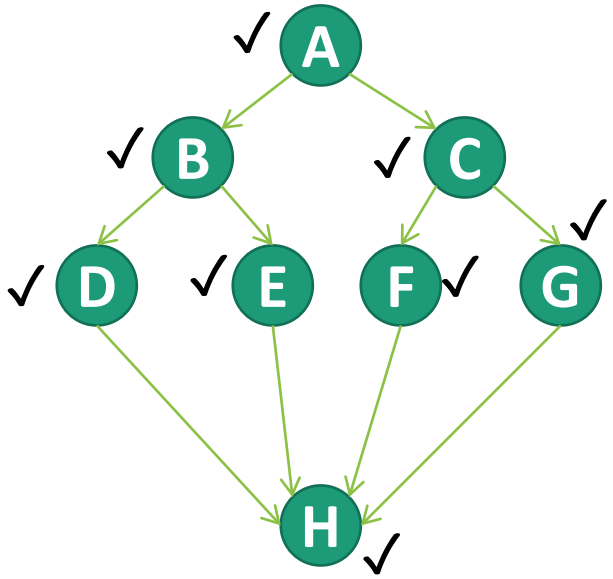
Introduction

- Search Techniques can be classified as:

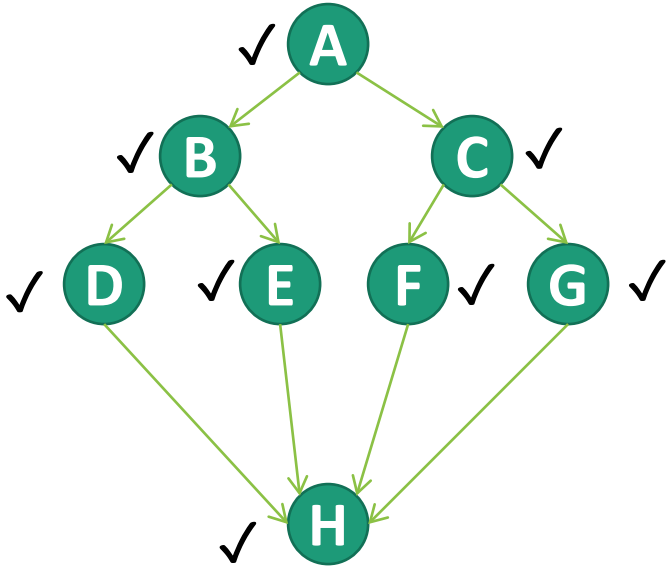
1. Uninformed/Blind Search Control Strategy:
   - Do not have additional information about states beyond problem definition.
   - Total search space is looked for the solution.
   - Example: Breadth First Search (BFS), Depth First Search (DFS), Depth Limited Search (DLS).

2. Informed/Directed Search Control Strategy:
   - Some information about problem space is used to compute the preference among various possibilities for exploration and expansion.
   - Examples: Best First Search, Problem Decomposition, A*, Mean end Analysis

Uninformed Search Techniques



Breadth First Search



Depth First Search

Breadth First Search Algorithm

Algorithm: Breadth-First Search

1. Create a variable called NODE-LIST and set it to all the 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

Algorithm: 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 the 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.

Difference Between DFS & BFS

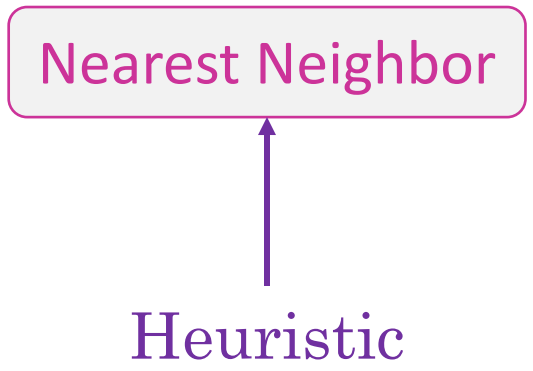
Depth First Search	Breath First Search
DFS requires less memory since only the nodes on the current path are stored.	BFS guarantees that the space of possible moves is systematically examined; this search requires considerably more memory resources.
By chance, DFS may find a solution without examining much of the search space at all. Then it finds a solution faster.	The search systematically proceeds testing each node that is reachable from a parent node before it expands to any child of those nodes.
If the selected path does not reach to the solution node, DFS gets stuck into a blind alley.	BFS will not get trapped exploring a blind alley.
Does not guarantee to find solution. Backtracking is required if wrong path is selected.	If there is a solution, BFS is guaranteed to find it.

Heuristic Search Techniques

- Every search process can be viewed as a traversal of a directed graph, in which the nodes represent problem states and the arcs represent relationships between states.
- The search process must find a path through this graph, starting at an initial state and ending in one or more final states.
- Domain-specific knowledge must be added to improve search efficiency.
- The Domain-specific knowledge about the problem includes the nature of states, cost of transforming from one state to another, and characteristics of the goals.
- This information can often be expressed in the form of Heuristic Evaluation Function.

Traveling Salesmen Problem (TSP)

- Start with any random city.
- Go to the next nearest city.



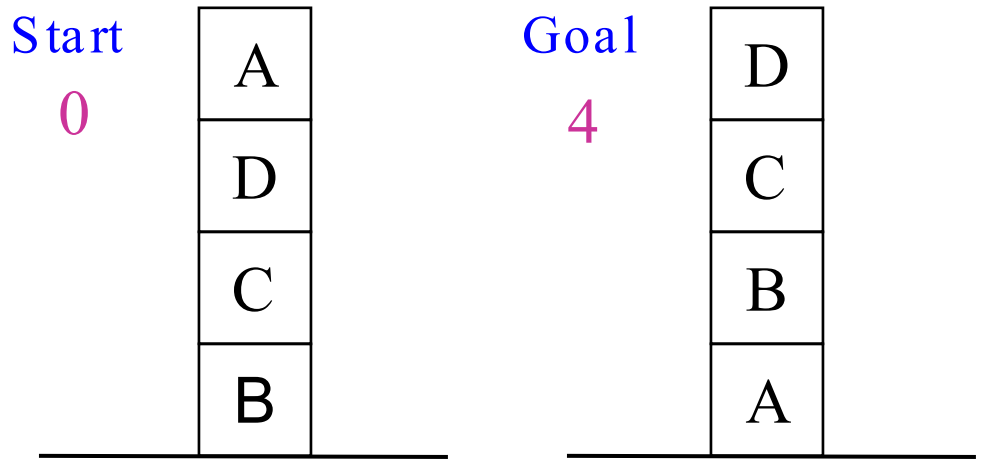
Heuristic Search Techniques

- Heuristic function maps from problem state descriptions to the measures of desirability, usually represented as numbers.
- The value of the heuristic function at a given node in the search process gives a good estimate of whether that node is on the desired path to a solution.
- Well-designed heuristic functions can play an important role in efficiently guiding a search process toward a solution.
- In general, heuristic search improves the quality of the path that is explored.
- In such problems, the search proceeds using current information about the problem to predict which path is closer to the goal and follow it, although it does not always guarantee to find the best possible solution.
- Such techniques help in finding a solution within reasonable time and space (memory).

Heuristic Search Techniques

- Some prominent intelligent search algorithms are stated below:
  1. Hill Climbing
  2. Best-first Search
  3. A* Search
  4. Constraint Search
  5. Means-ends analysis

Hill Climbing Example - Blocks World Problem

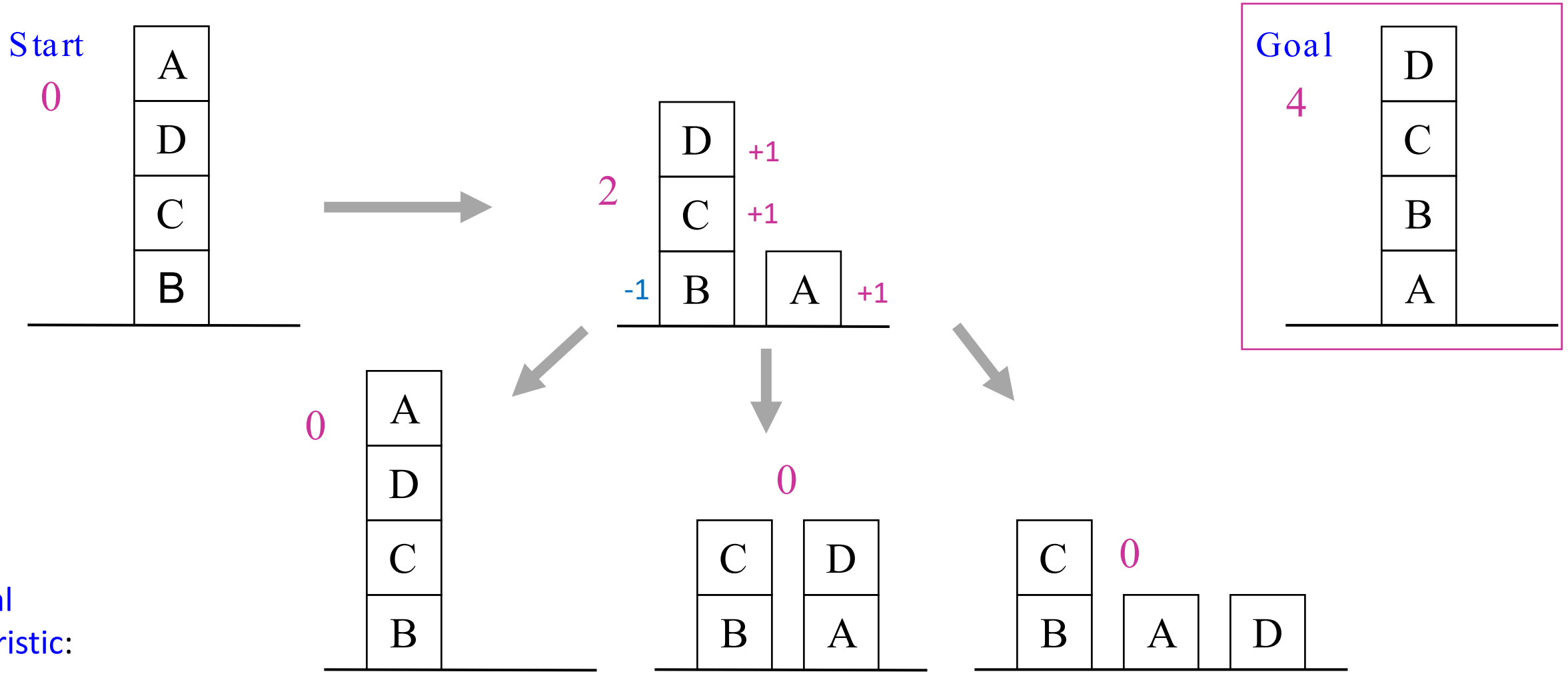


Local heuristic:

+1 for each block that is resting on the thing it is supposed to be resting on.

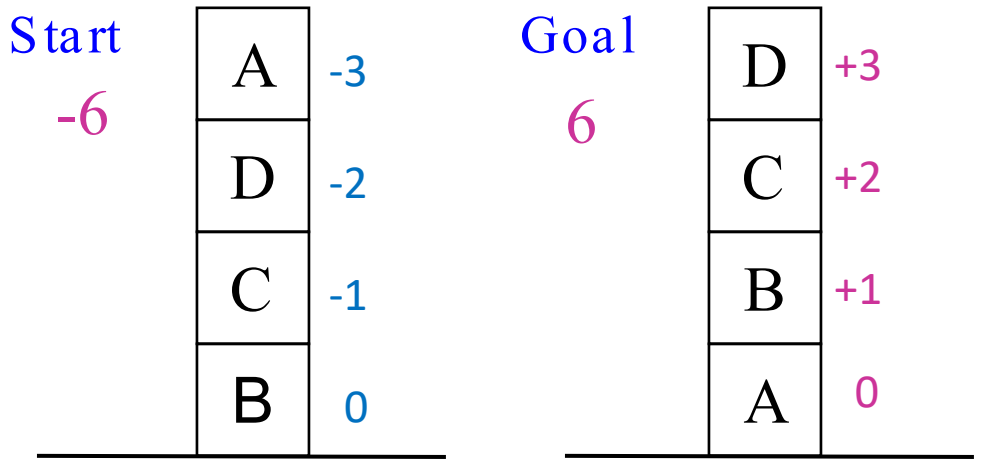
-1 for each block that is resting on a wrong thing.

Hill Climbing Example - Blocks World Problem



Local heuristic:

Hill Climbing Example - Blocks World Problem



Global heuristic:

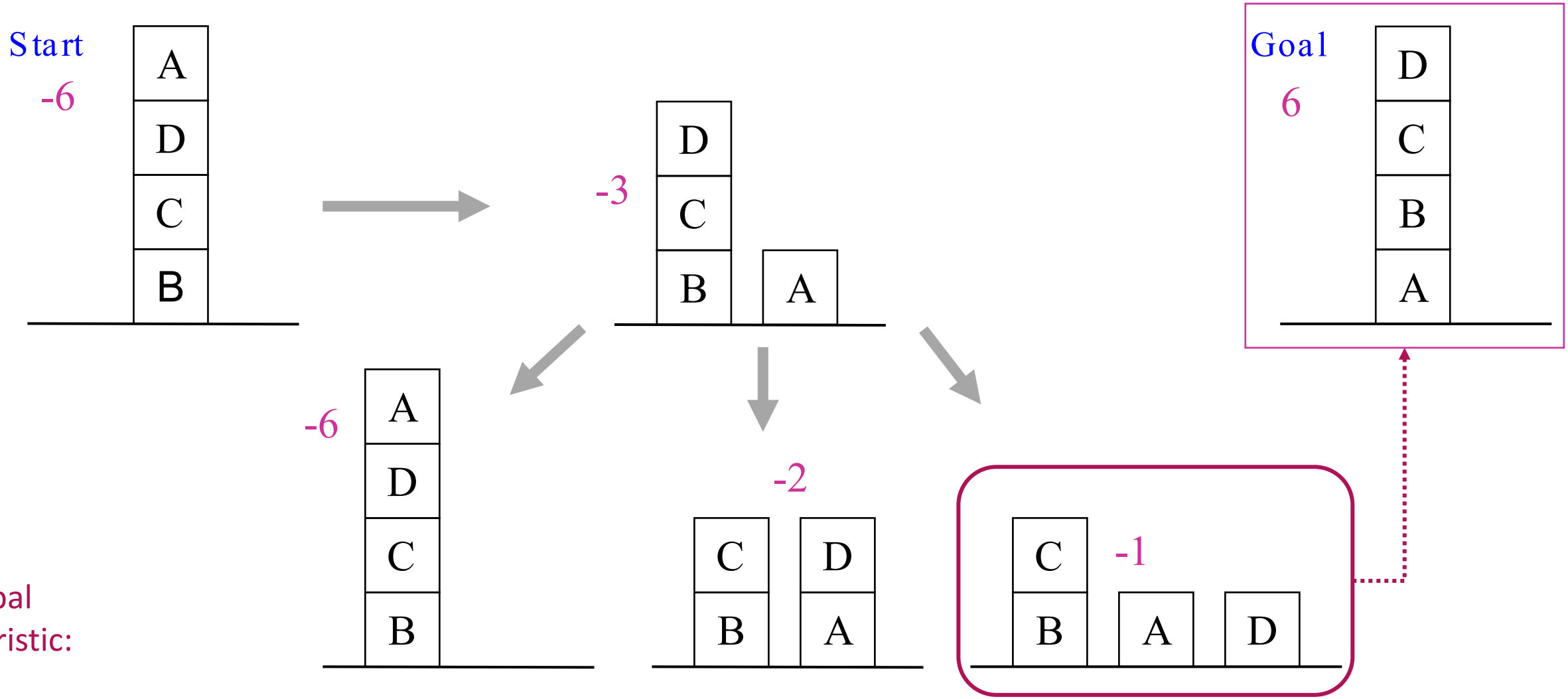
For each block that has the correct support structure:

+1 to every block in the support structure.

For each block that has a wrong support structure:

-1 to every block in the support structure.

Hill Climbing Example - Blocks World Problem



Simple Hill Climbing - Algorithm

1. Evaluate the initial state. If it is also goal state, then return it and quit. Otherwise continue with the initial state as the current state.
2. Loop until a solution is found or until there are no new operators left to be applied in the current state:
   a. Select an operator that has not yet been applied to the current state and apply it to produce a new state.
   b. Evaluate the new state
      i. If it is the goal state, then return it and quit.
      ii. If it is not a goal state but it is better than the current state, then make it the current state.
      iii. If it is not better than the current state, then continue in the loop.

Drawbacks of Hill Climbing

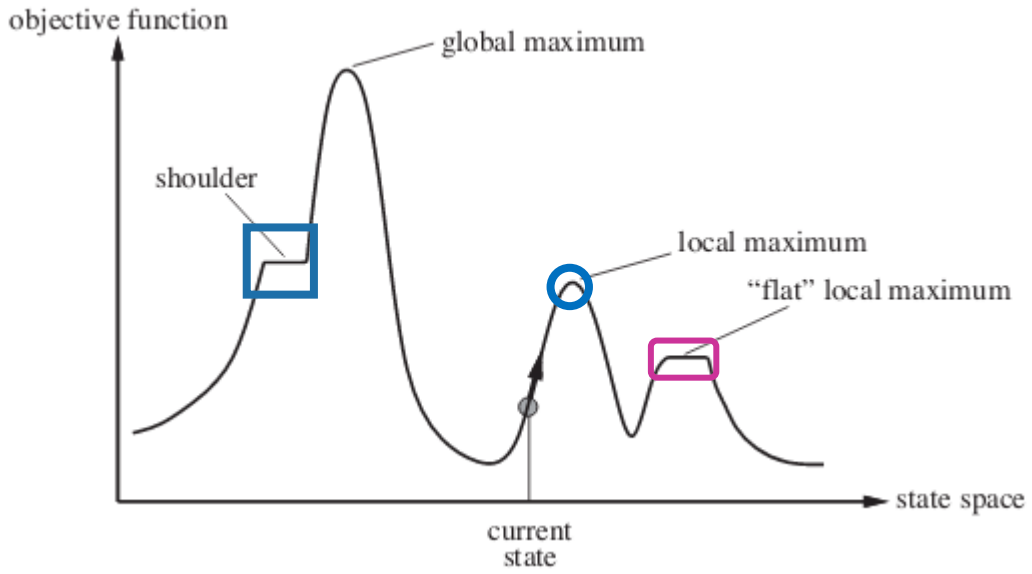


Image source: https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/

- Local Maxima: a local maximum is a state that is better than all its neighbors but is not better than some other states further away.
- To overcome local maximum problem: Utilize backtracking technique. Maintain a list of visited states and explore a new path.

- Plateau: a plateau is a flat area of the search space in which, a whole set of neighboring states have the same values.
- To overcome plateaus: Make a big jump. Randomly select a state far away from current state.

- Ridge: is a special kind of local maximum. It is an area of the search space that is higher than surrounding areas and that itself has slop.
- To overcome Ridge: In this kind of obstacle, use two or more rules before testing. It implies moving in several directions at once.

Steepest-Ascent Hill Climbing - Algorithm

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or until a complete iteration produces no change to current state:
   a. Let S be a state such that any possible successor of the current state will be better than S.
   b. For each operator that applies to the current state do:
      - Apply the operator and generate a new state
      - Evaluate the new state. If it is a goal state, then return it and quit. If not, compare it to S. If it is better, then set S to this state. If it is not better, leave S alone.
   c. If the S is better than the current state, then set current state to S.

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.

Best First Search

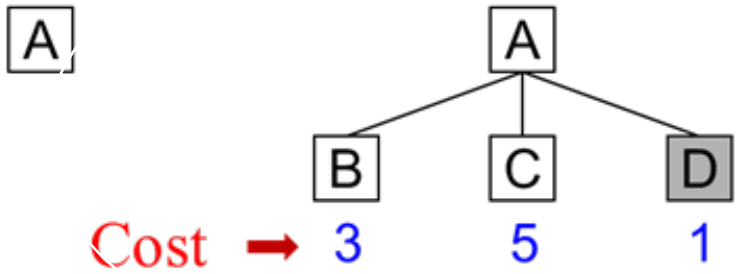
- DFS is good because it allows a solution to be found without expanding all competing branches. BFS is good because it does not get trapped on dead end paths.
- Best first search combines the advantages of both DFS and BFS into a single method.
- One way of combining BFS and DFS is to follow a single path at a time, but switch paths whenever some competing path looks more promising than the current one does.
- At each step of the Best First Search process, we select the most promising of the nodes we have generated so far.
- This is done by applying an appropriate heuristic function to each of them.
- We then expand the chosen node by using the rules to generate its successors.
- If one of them is a solution, we can quit. If not, all those new nodes are added to the set of nodes generated so far.

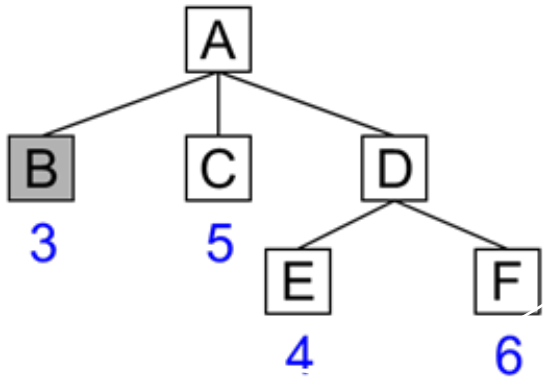
Best First Search

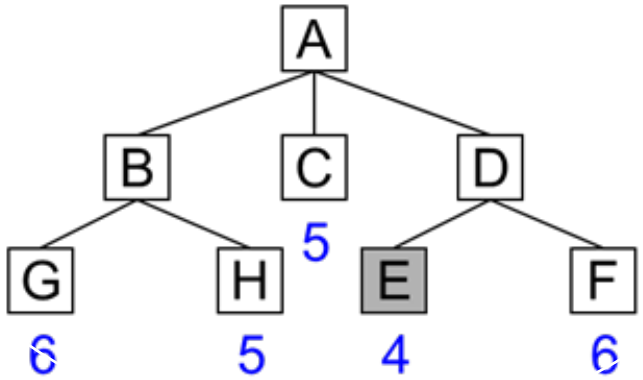
- Algorithm: Best First Search

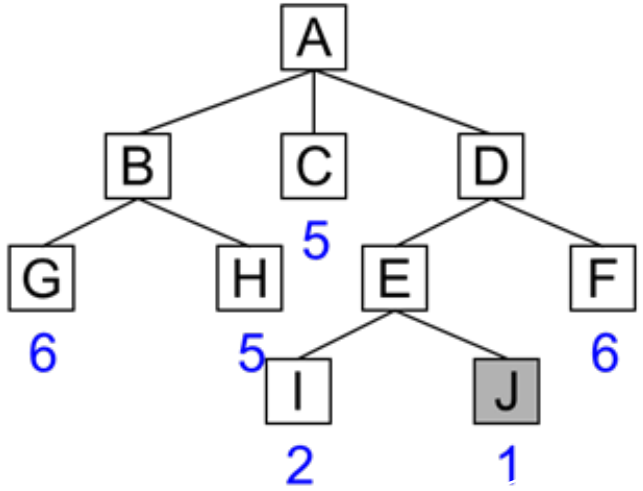
1. Start with OPEN containing just the initial state
2. Until a goal is found or there are no nodes left on OPEN do:
   a. Pick the best node on OPEN
   b. Generate its successors.
   c. For each successor do:
      I. If it has not been generated before, evaluate it, add it to OPEN and record its parent.
      II. If it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have.

Best First Search







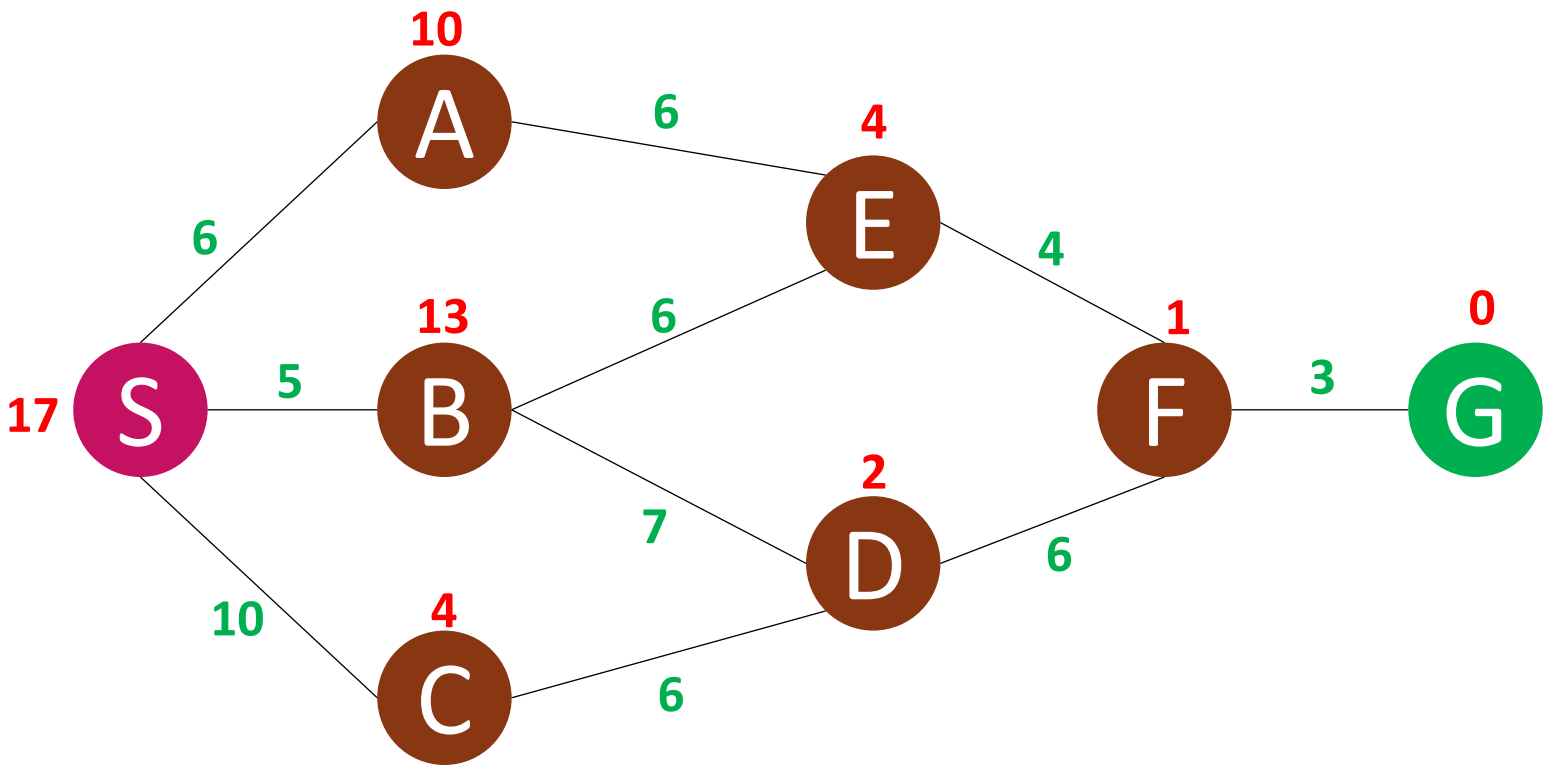


A* Example

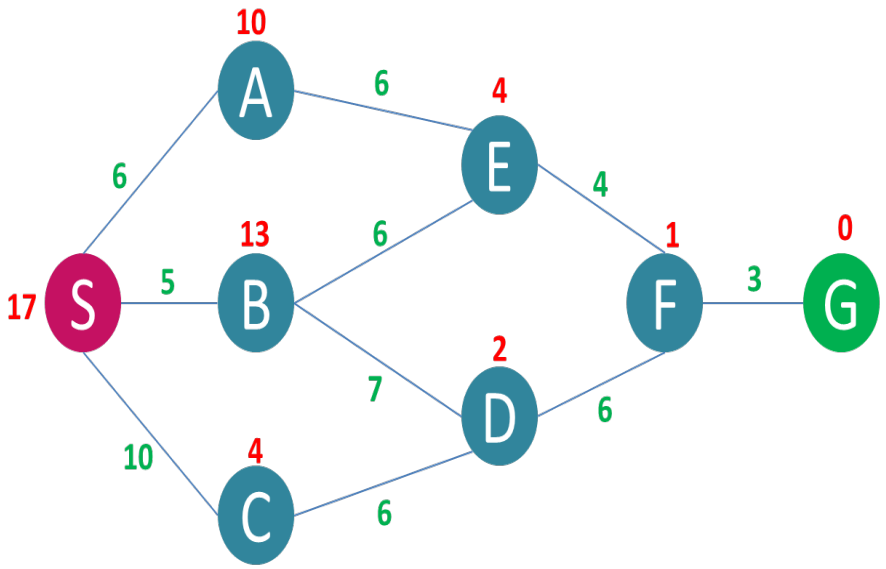
$f'(n) = g(n) + h'(n)$

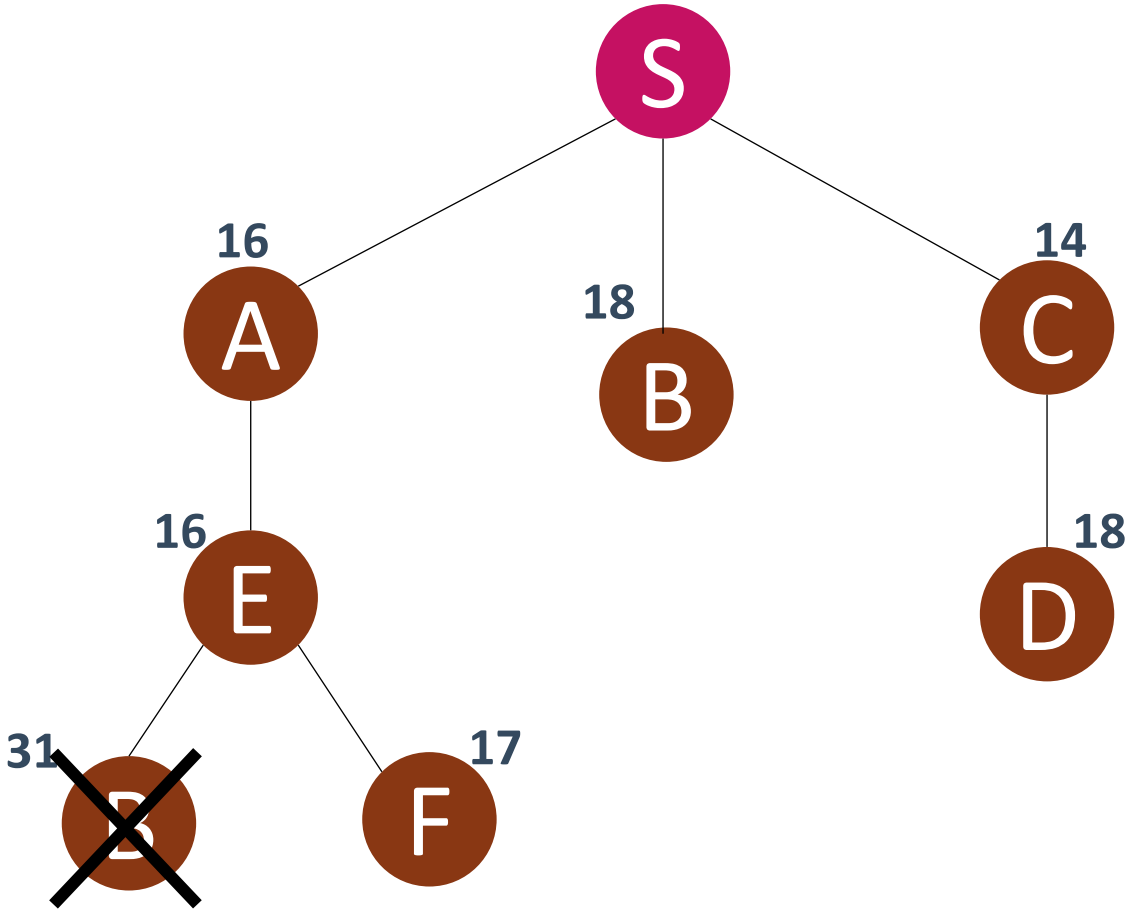
h'(n): The function h' is an estimate of the additional cost of getting from the current node to a goal state.

g(n) : The function g is a cost of getting from initial state to the current node.

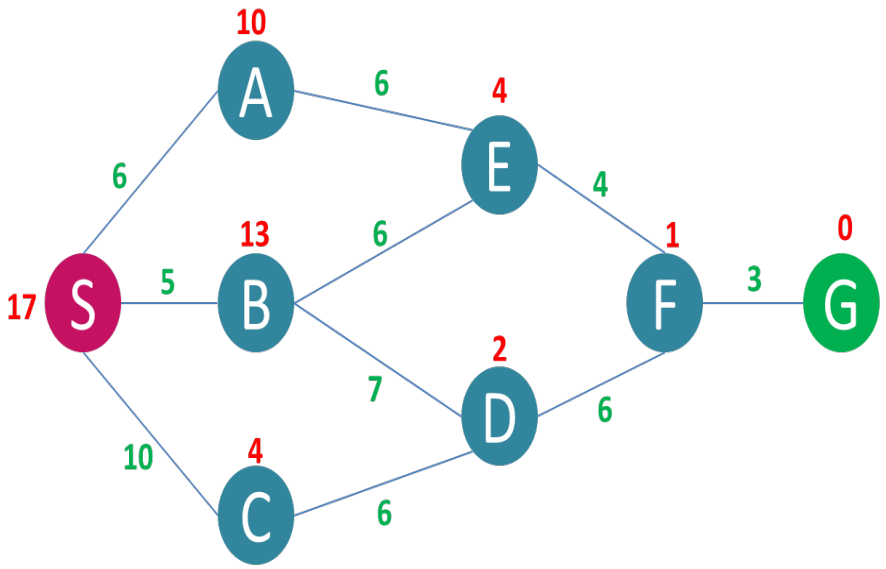


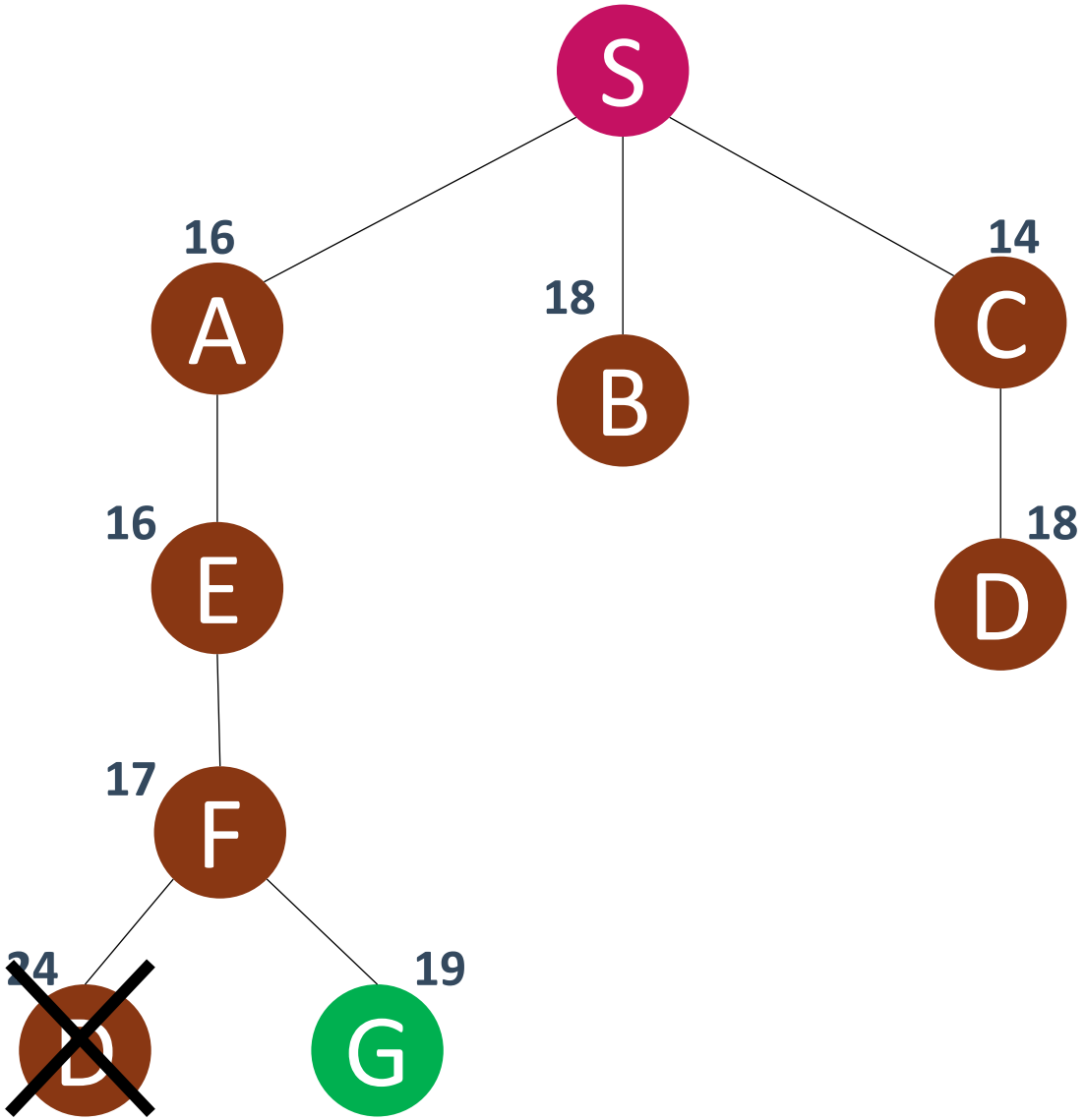
A* Example


10
A
6
4
E
6
4
13
6
1
0
17
S
5
B
F
3
G
2
7
D
6
10
4
C
6

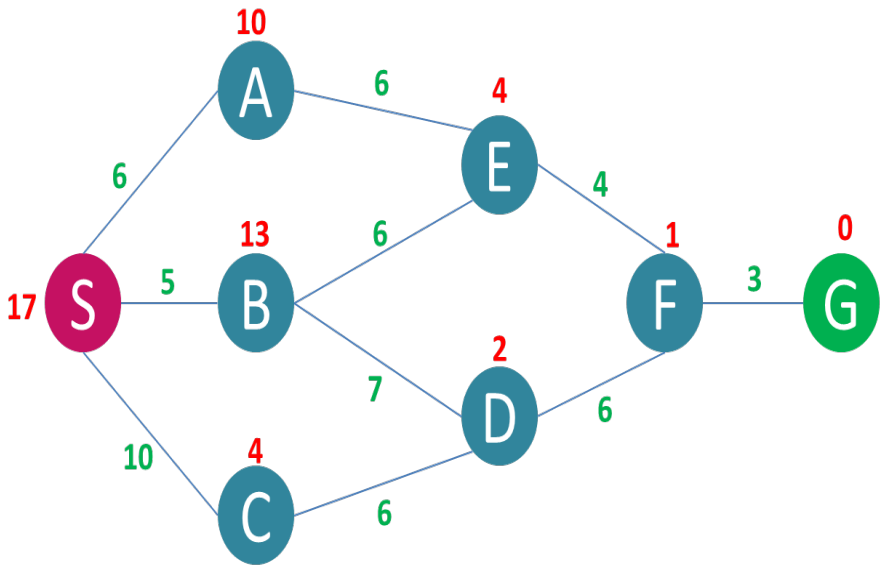

S
16
18
14
A
B
C
16
18
E
D
31
17
B
F

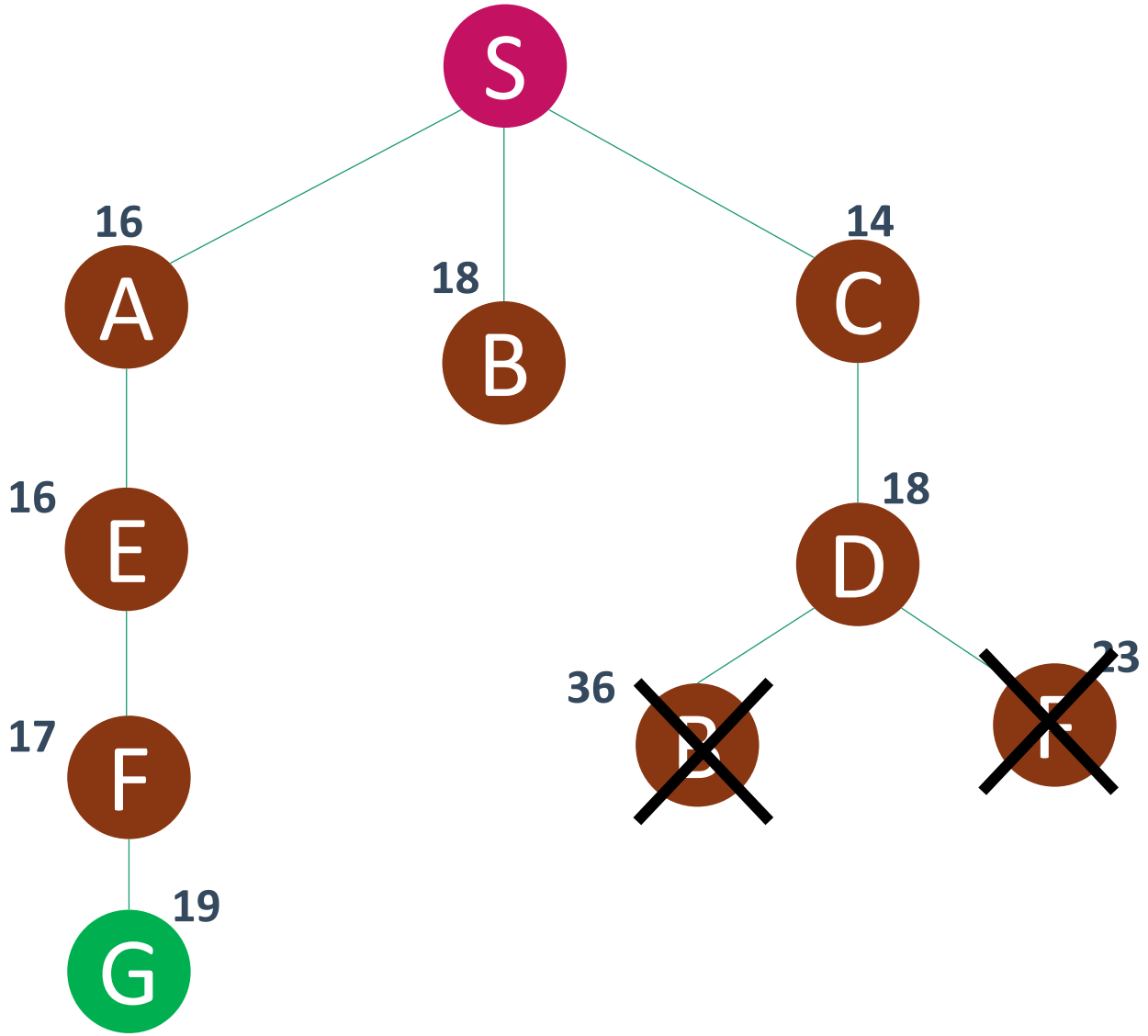
A* Example


10
A
6
4
E
6
4
13
6
1
0
17
S
5
B
F
3
G
2
7
D
6
10
4
C
6

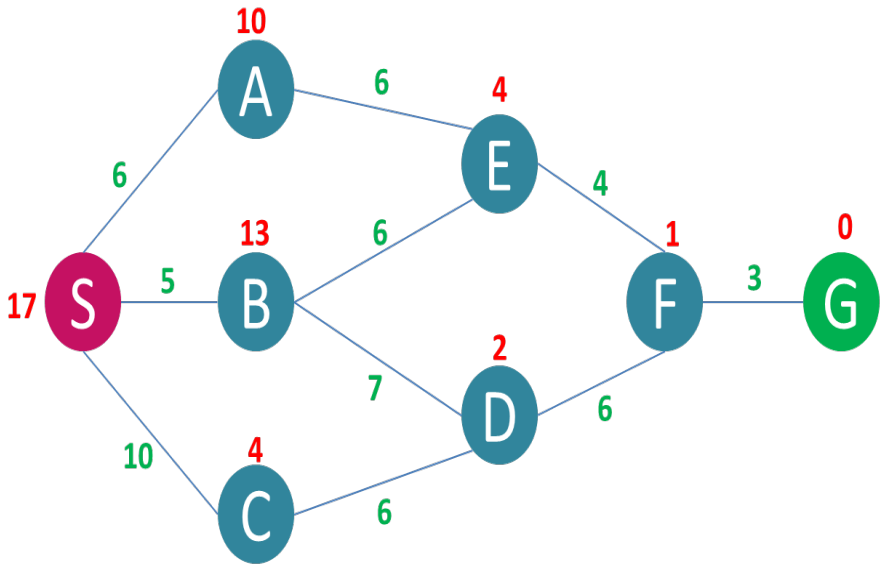

S
16
18
14
A
B
C
16
18
E
D
17
F
24
19
D
G

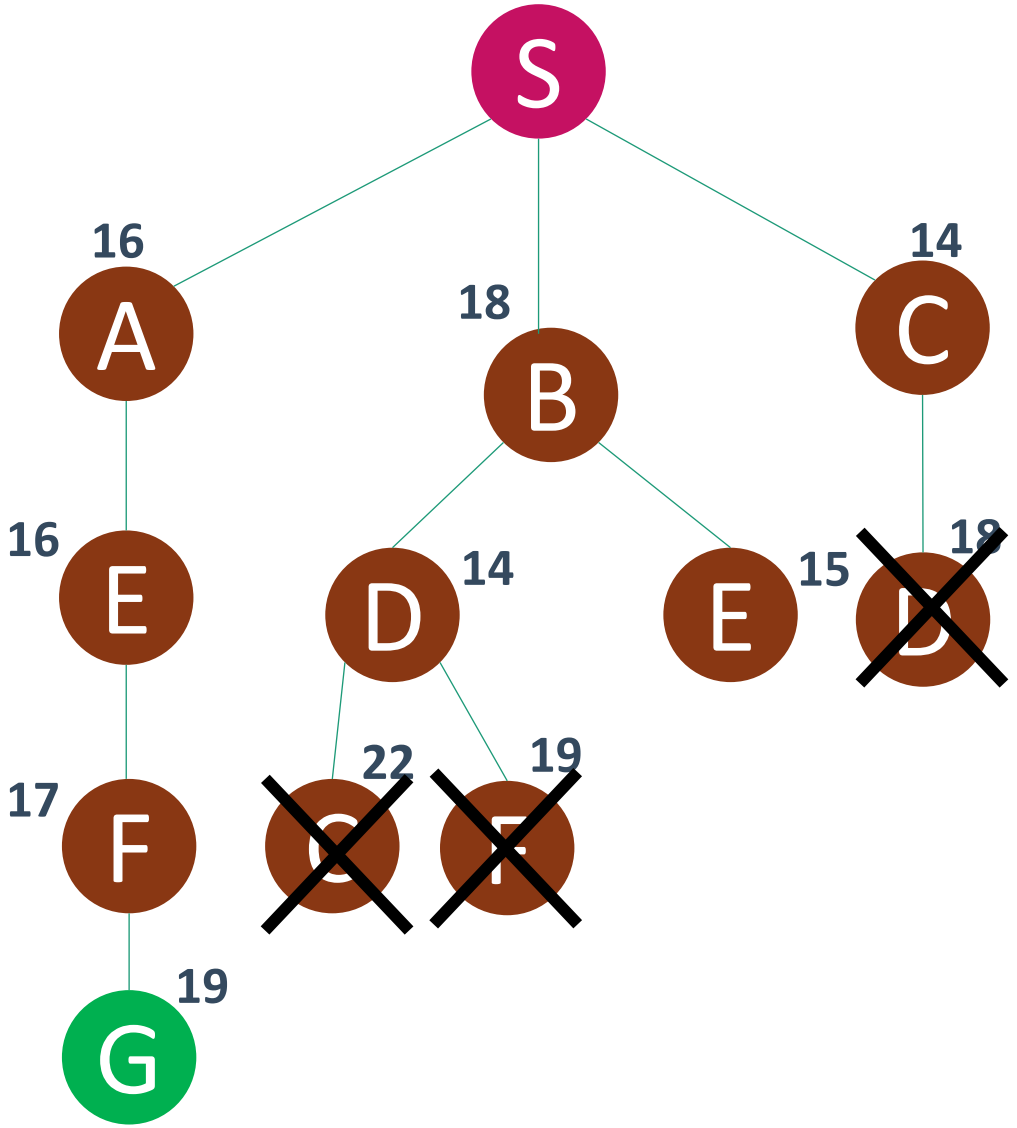
A* Example


10
A
6
4
E
6
4
13
6
1
0
17
S
5
B
F
3
G
2
7
D
6
10
4
C
6

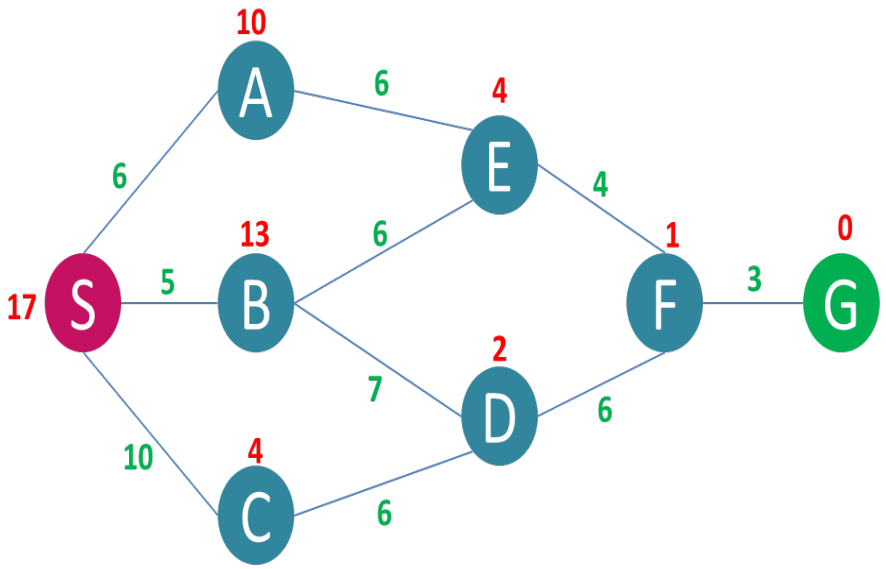

S
16
18
14
A
B
C
16
18
E
D
36
23
17
F
19
G

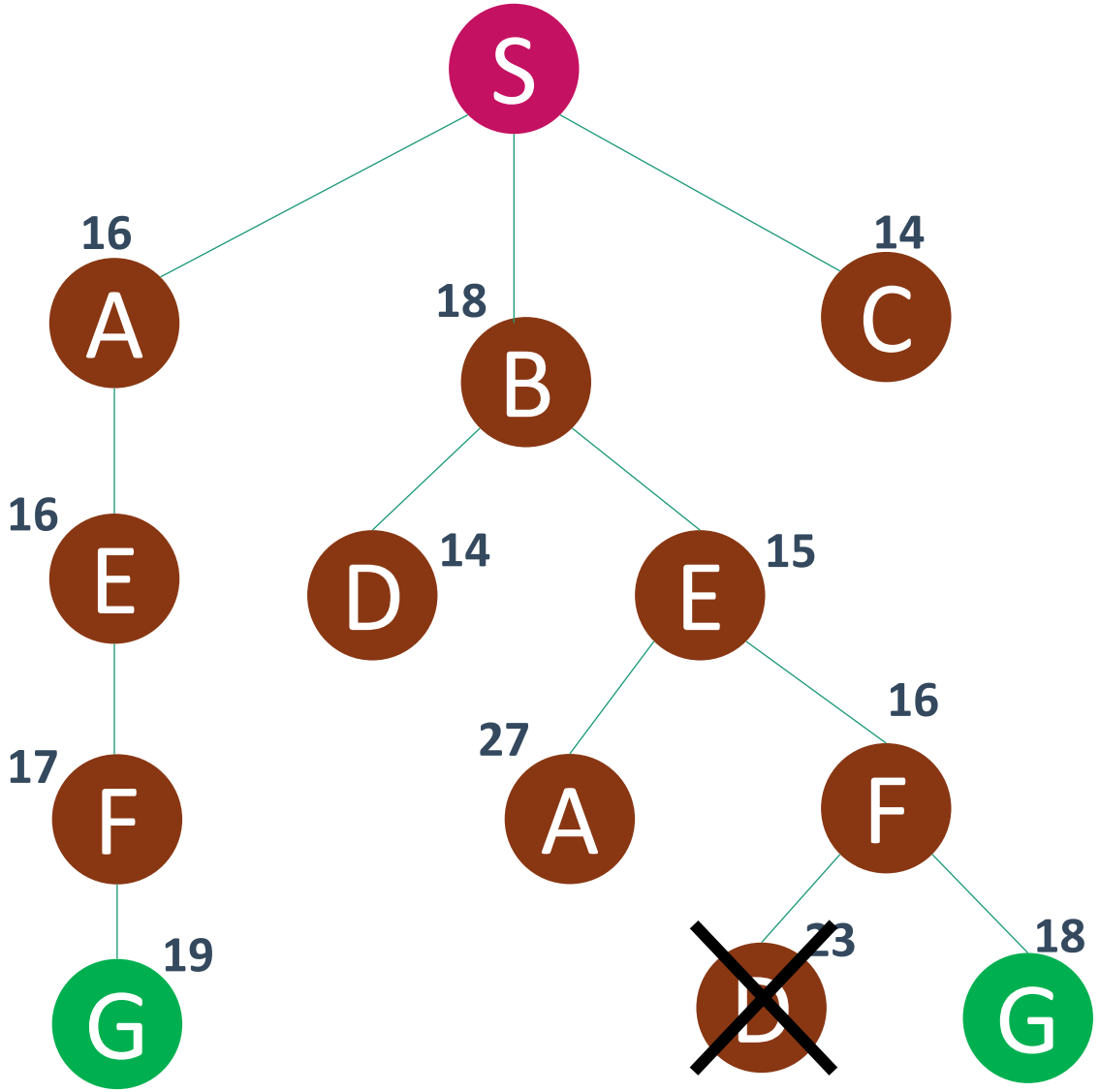
A* Example


10
A
6
4
E
6
4
13
6
1
0
17
S
5
B
F
3
G
2
7
D
6
10
4
C
6

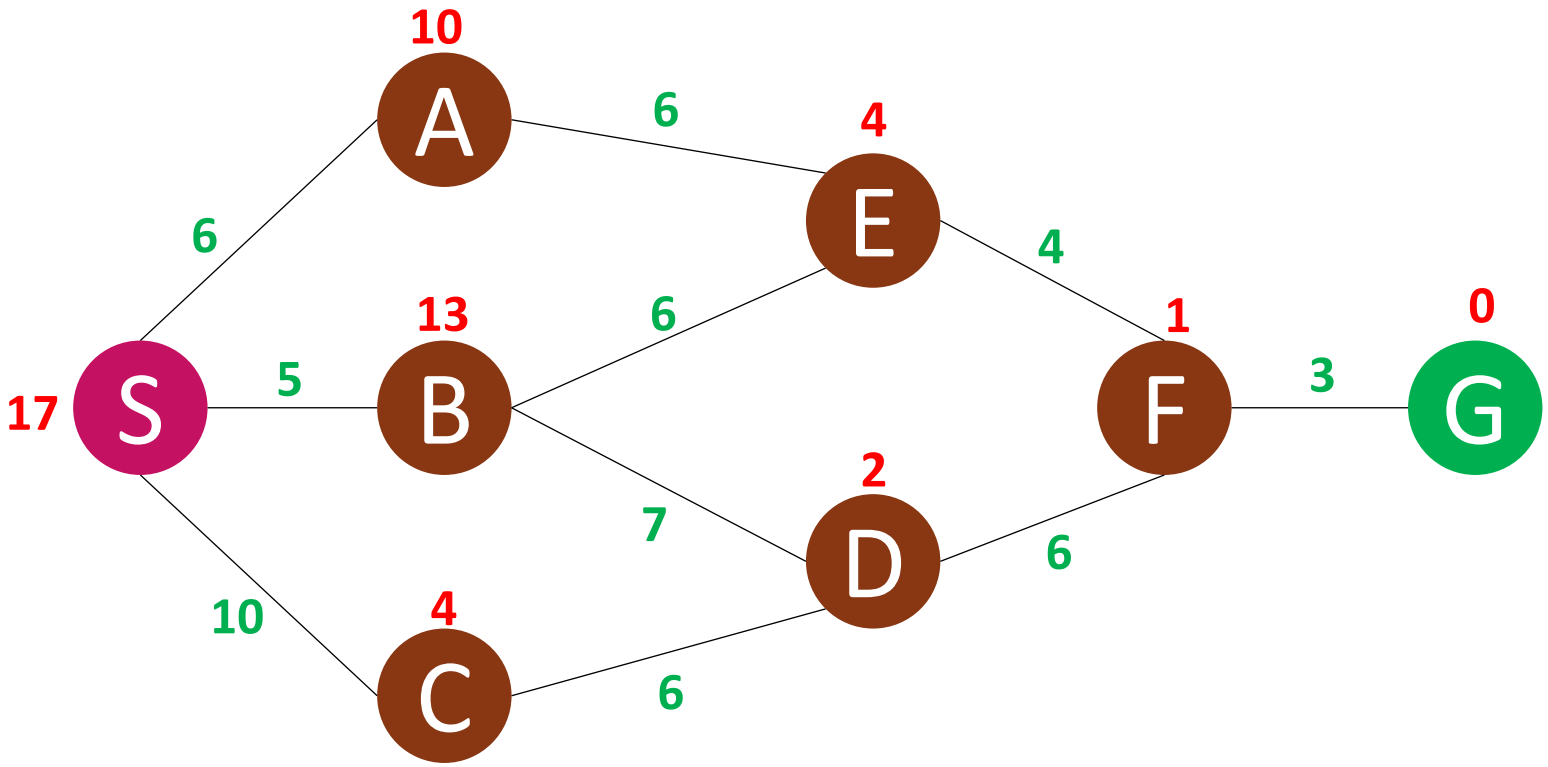

S
16
A
18
B
14
C
16
E
D
14
E
15
19
D
22
19
17
F
G
19

A* Example


10
A
6
4
E
6
4
13
6
1
0
17
S
5
B
F
3
G
2
7
D
6
10
4
C
6


S
16
A
18
B
14
C
16
E
D
14
E
15
17
F
27
A
16
F
19
G
23
D
18
G

A* Example Solution



$S \rightarrow B \rightarrow E \rightarrow F \rightarrow G$

Total Cost = 18

The A* Algorithm

- This algorithm uses following functions:
  1. **f':** Heuristic function that estimates the merits of each node we generate. f' represents an estimate of the cost of getting from the initial state to a goal state along with the path that generated the current node. f' = g + h'
  2. **g:** The function g is a measure of the cost of getting from initial state to the current node.
  3. **h':** The function h' is an estimate of the additional cost of getting from the current node to a goal state.

- The algorithm also uses the lists: OPEN and CLOSED
  - **OPEN** - nodes that have been generated and have had the heuristic function applied to them but which have not yet been examined. OPEN is actually a priority queue in which the elements with the highest priority are those with the most promising value of the heuristic function.
  - **CLOSED** - nodes that have already been examined. We need to keep these nodes in memory if we want to search a graph rather than a tree, since whenever a new node is generated; we need to check whether it has been generated before.

The A* Algorithm

Step 1:

- Start with OPEN containing only initial node.
- Set that node's g value to 0, its h' value to whatever it is, and its f' value to h'+0 or h'.
- Set CLOSED to empty list.

Step 2: Until a goal node is found, repeat the following procedure:

- If there are no nodes on OPEN, report failure.
- Otherwise select the node on OPEN with the lowest f' value.
- Call it BESTNODE. Remove it from OPEN. Place it in CLOSED.
- See if the BESTNODE is a goal state. If so exit and report a solution.
- Otherwise, generate the successors of BESTNODE but do not set the BESTNODE to point to them yet.

The A* Algorithm

Step 3: For each of the SUCCESSOR, do the following:

a. Set SUCCESSOR to point back to BESTNODE. These backwards links will make it possible to recover the path once a solution is found.

b. Compute g(SUCCESSOR) = g(BESTNODE) + the cost of getting from BESTNODE to SUCCESSOR

c. See if SUCCESSOR is the same as any node on OPEN. If so call the node OLD.

   i. Check whether it is cheaper to get to OLD via its current parent or to SUCESSOR via BESTNODE by comparing their g values.

   ii. If OLD is cheaper, then do nothing. If SUCCESSOR is cheaper then reset OLD's parent link to point to BESTNODE.

   iii. Record the new cheaper path in g(OLD) and update f'(OLD).

   iv. If SUCCESSOR was not on OPEN, see if it is on CLOSED. If so, call the node on CLOSED OLD and add OLD to the list of BESTNODE's successors.

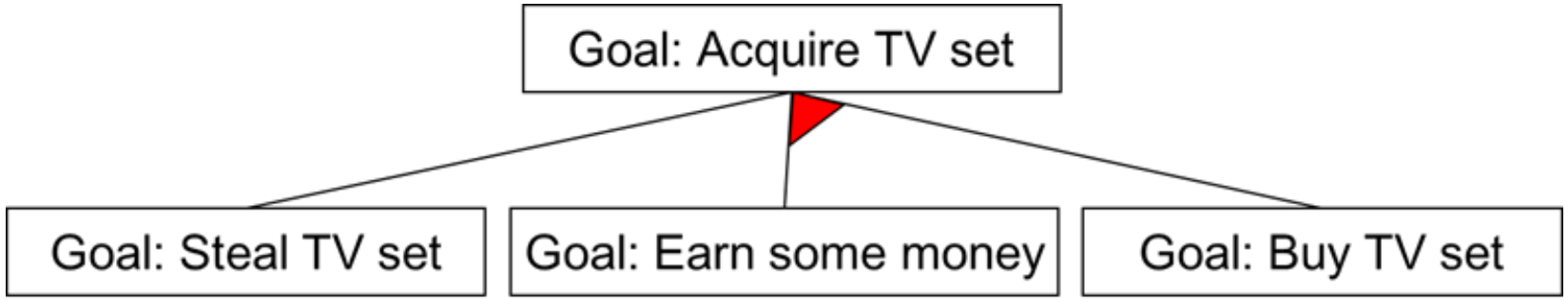
   v. If SUCCESSOR was not already on either OPEN or CLOSED, then put it on OPEN and add it to the list of BESTNODE's successors. Compute f'(SUCCESSOR) = g(SUCCESSOR) + h'(SUCCESSOR)

Problem Reduction

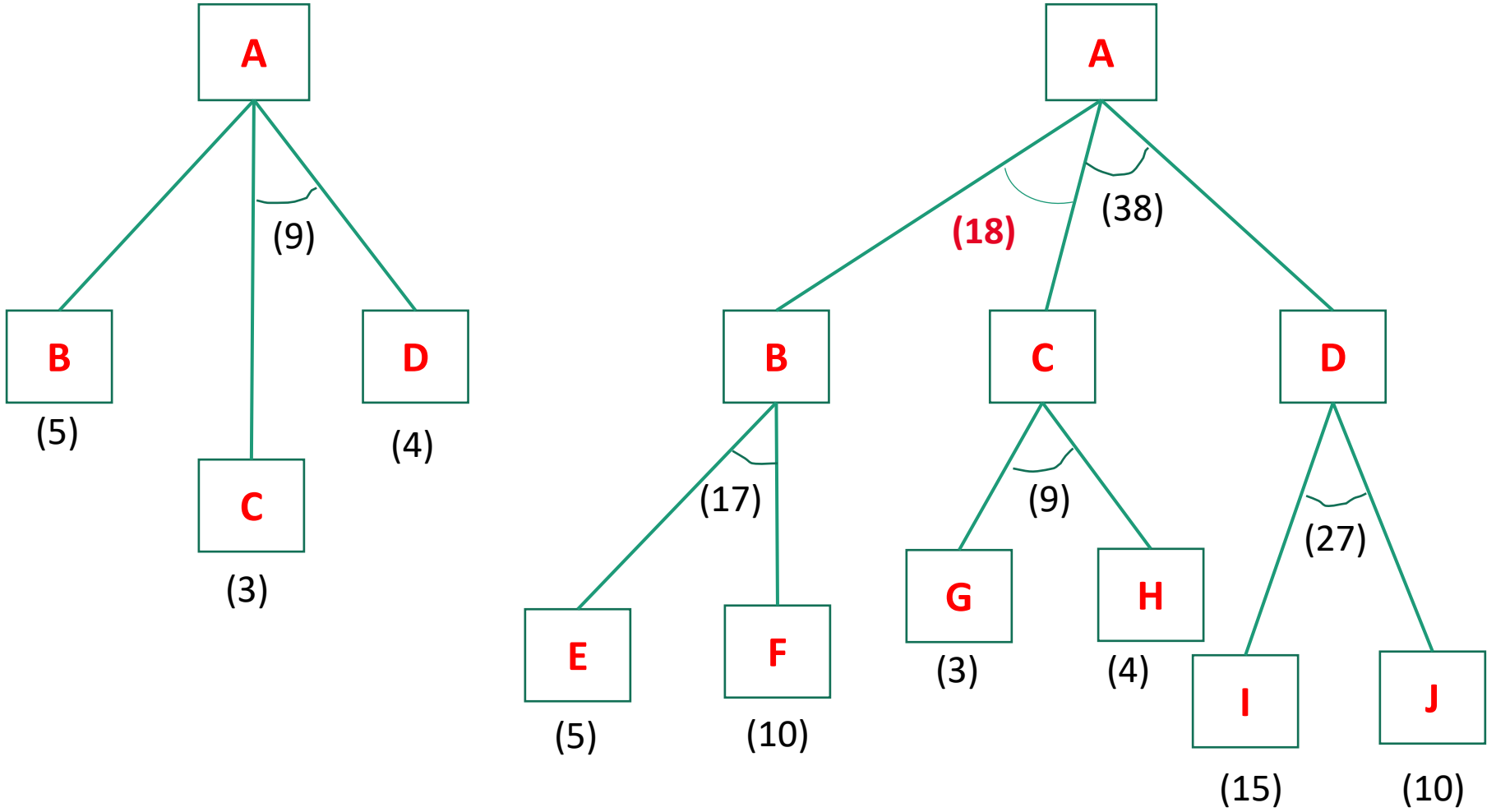
- AND-OR graph (or tree) is useful for representing the solution of problems that can be solved by decomposing them into a set of smaller problems, all of which must then be solved.
- This decomposition or reduction generates arcs that we call AND arcs.
- One AND arc may point to any numbers of successor nodes. All of which must then be solved in order for the arc to point solution.
- In order to find solution in an AND-OR graph we need an algorithm similar to best –first search but with the ability to handle the AND arcs appropriately.
- We define FUTILITY, if the estimated cost of solution becomes greater than the value of FUTILITY then we abandon the search, FUTILITY should be chosen to correspond to a threshold.

AND-OR Graph

- In following figure AND arcs are indicated with a line connecting all the components.



AND – OR Graph



Heuristic Search Techniques for AO*

- Traverse the graph starting at the initial node and following the current best path, and accumulate the set of nodes that are on the path and have not yet been expanded.
- Pick one of these unexpanded nodes and expand it. Add its successors to the graph and computer f'(cost of the remaining distance) for each of them.
- Change the f' estimate of the newly expanded node to reflect the new information produced by its successors. Propagate this change backward through the graph. Decide which is the current best path.
- The propagation of revised cost estimation backward is in the tree is not necessary in A* algorithm. This is because in AO* algorithm expanded nodes are re-examined so that the current best path can be selected.

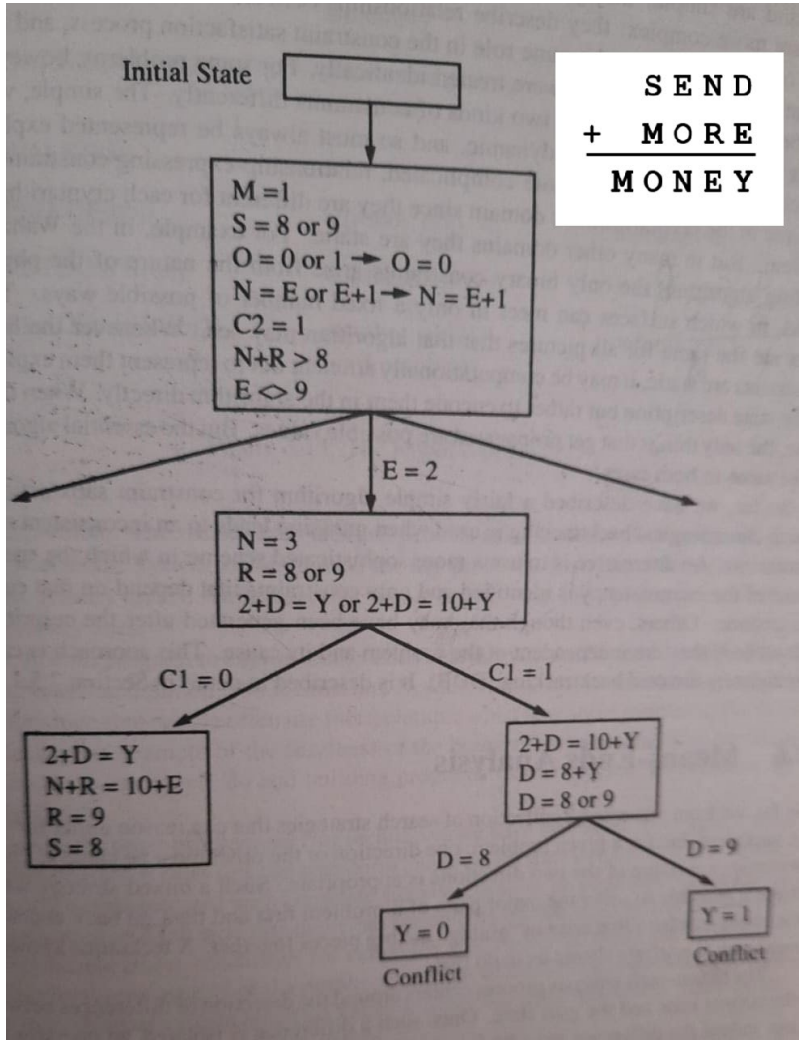
Constraint Satisfaction

- Many AI problems can be viewed as problems of constraint satisfaction.
- For example, Crypt-arithmetic puzzle:

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

- As compared with a straightforward search procedure, viewing a problem as one of the constraint satisfaction can substantially reduce the amount of search.
- Two-step process:
  1. Constraints are discovered and propagated as far as possible.
  2. If there is still not a solution, then search begins, adding new constraints.
- Initial state contains the original constraints given in the problem.
- A goal state is any state that has been constrained "enough". i.e. The goal is to discover some problem state that satisfies the given set of constraints.

Constraint Satisfaction

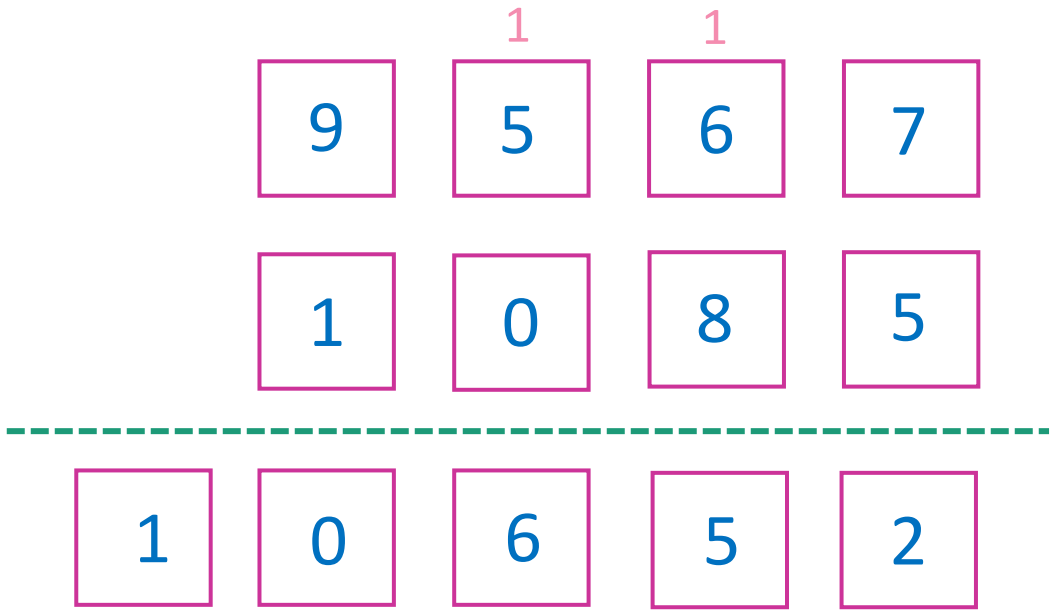


Crypt-arithmetic Puzzle

Initial state:

- Assign values between 0 to 9.
- No two letters have the same value.
- The sum of the digits must be as shown.

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



S	9
E	5
N	6
D	7
M	1
O	0
R	8
Y	2

Means End Analysis

- Most of the search strategies either reason forward of backward, Often a mixture of the two directions is appropriate.
- Such mixed strategy would make it possible to solve the major parts of problem first and solve the smaller problems the arise when combining them together.
- Such a technique is called Means - Ends Analysis.
- The means -ends analysis process centers around finding the difference between current state and goal state.
- The problem space of means - ends analysis has
  - an initial state and one or more goal state,
  - a set of operate with a set of preconditions their application and difference functions that computes the difference between two state a(i) and s(j).

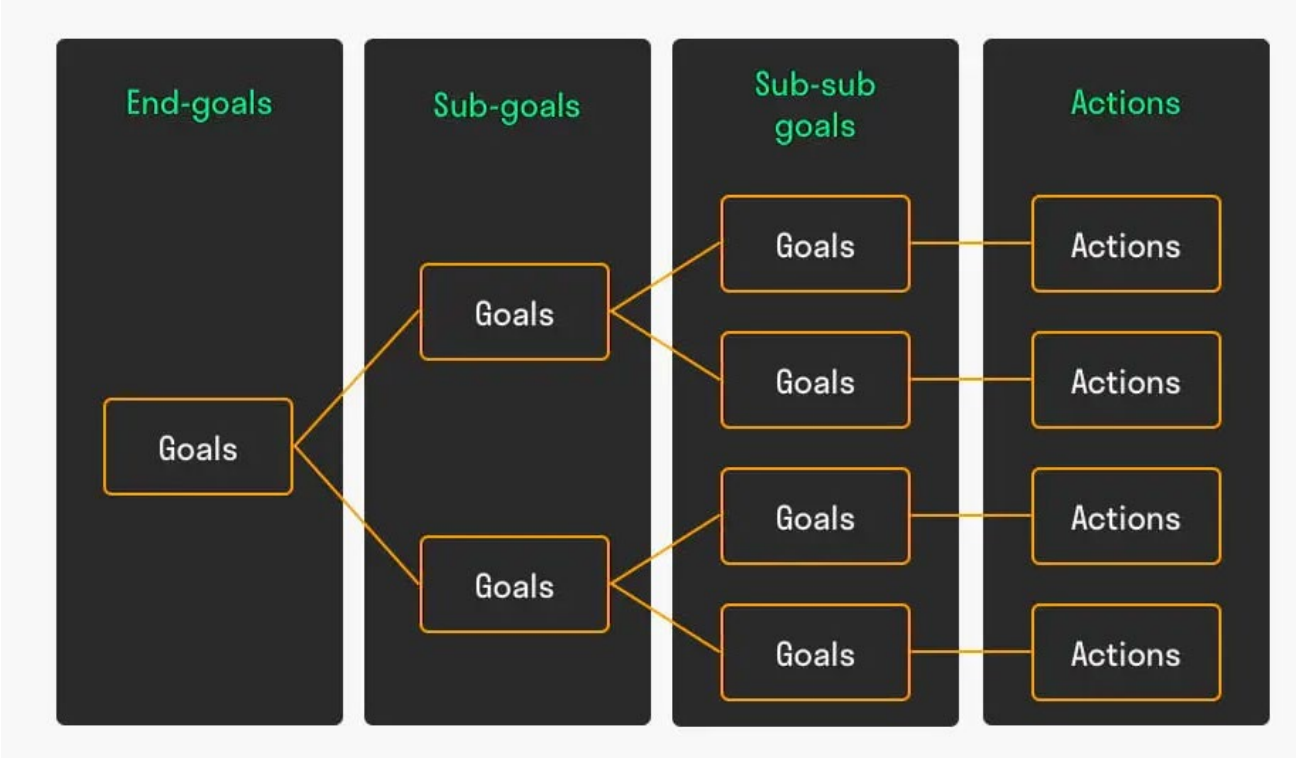
Means End Analysis

- The means-ends analysis process can be applied recursively for a problem.
- Following are the main Steps which describes the working of MEA technique for solving a problem.
  - First, evaluate the difference between Initial State and final State.
  - Select the various operators which can be applied for each difference.
  - Apply the operator at each difference, which reduces the difference between the current state and goal state.

- In the MEA process, we detect the differences between the current state and goal state.
- Once these differences occur, then we can apply an operator to reduce the differences.
- But sometimes it is possible that an operator cannot be applied to the current state.

Means End Analysis

- So, we create the sub-problem of the current state, in which operator can be applied, such type of backward chaining in which operators are selected, and then sub goals are set up to establish the preconditions of the operator is called Operator Subgoaling.



Algorithm : Means-Ends Analysis

1. Compare CURRENT to GOAL, if there are no differences between both then return Success and Exit.
2. Else, select the most significant difference and reduce it by doing the following steps until the success or failure occurs.
   a. Select a new operator O which is applicable for the current difference, and if there is no such operator, then signal failure.
   b. Attempt to apply operator O to CURRENT. Make a description of two states.
      i. O-Start, a state in which O?s preconditions are satisfied.
      ii. O-Result, the state that would result if O were applied In O-start.
   c. If (First-Part <------ MEA (CURRENT, O-START) And (LAST-Part <----- MEA (O-Result, GOAL), are successful, then signal Success and return the result of combining FIRST-PART, O, and LAST-PART.

Thank You!

Unit: 2
AI Problems and Search

Outline

- Overview
- MiniMax Search Procedure
- Alpha-Beta Cut-offs
- Iterative deepening

Introduction

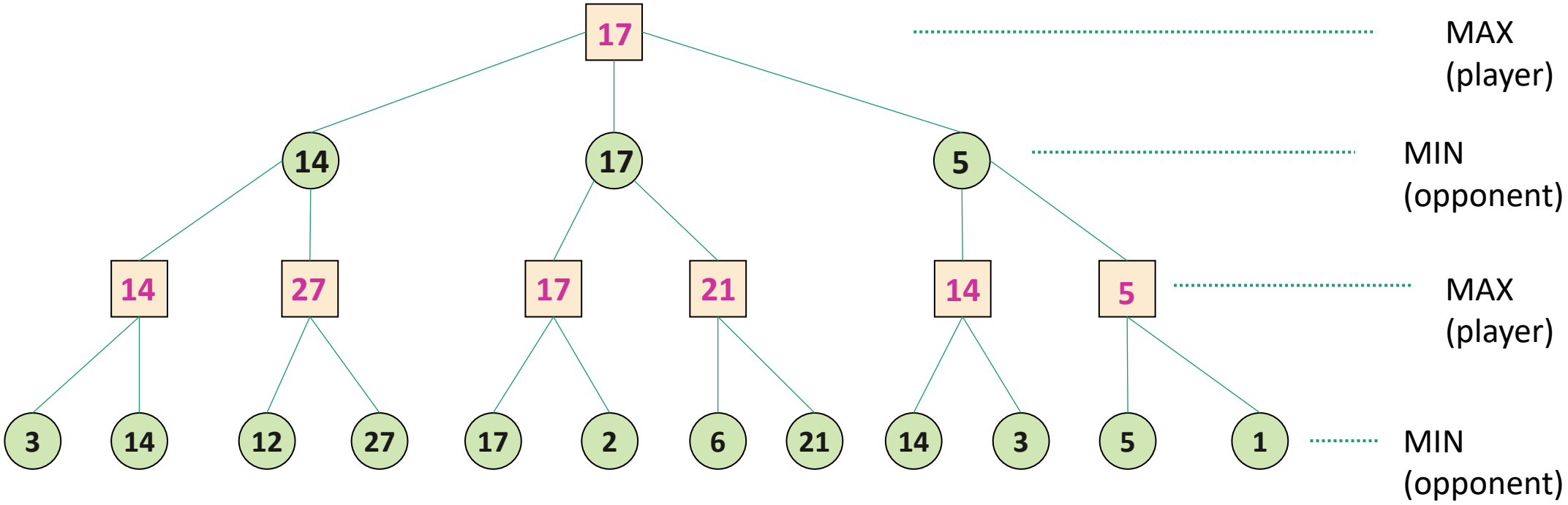
- Game Playing is an important domain of Artificial Intelligence.
- There are two reasons that games appeared to be a good domain.
  1. They provide a structured task in which it is very easy to measure success or failure.
  2. They are easily solvable by a straightforward search from the starting state to a winning position.
- Games require only the domain knowledge such as the rules, legal moves and the conditions of winning or losing the game.
- In a two-player game, both the players try to win the game. So, both of them try to make the best move possible at each turn.
- To improve the effectiveness of a search based problem solving program two things can be done.
  1. Improve generate procedure so that only good moves are generated.
  2. Improve test procedure so that the best move will be recognized and explored first.

Introduction

- If we use legal-move generator then the test procedure will have to look at each of them, because the test procedure must look at so many possibilities and it must be fast.
- The depth of the resulting tree or graph and its branching factor will be too large.
- Instead of legal-move generator we can use plausible-move generator in which only some small numbers of promising moves are generated.
- As the number of legal available moves increases it becomes increasingly important in applying heuristics to select only those moves that seem more promising.
- The performance of overall system can be improved by adding heuristic knowledge into both the generator and the tester.
- It is possible to search tree only ten or twenty moves deep then in order to choose the best move, the resulting board positions must be compared to discover which is most advantageous.

Introduction

- This is done using static evaluation function, which uses whatever information it has to evaluate individual board position by estimating how likely they are to lead eventually to a win.
- The most common search technique in game playing is Minimax search procedure.



The MINIMAX Search Procedure

- The Minimax search is a depth first and depth limited procedure.
- The idea is to start at the current position and use the plausible-move generator to generate the set of possible successor positions.
- Now we can apply the static evaluation function to those positions and simply choose the best one.
- After doing so, we can back that value up to the starting position.
- Here, we assume that static evaluation function returns larger values to indicate good situations for us.
- So our goal is to maximize the value of the static evaluation function of the next board position.
- The opponents' goal is to minimize the value of the static evaluation function.

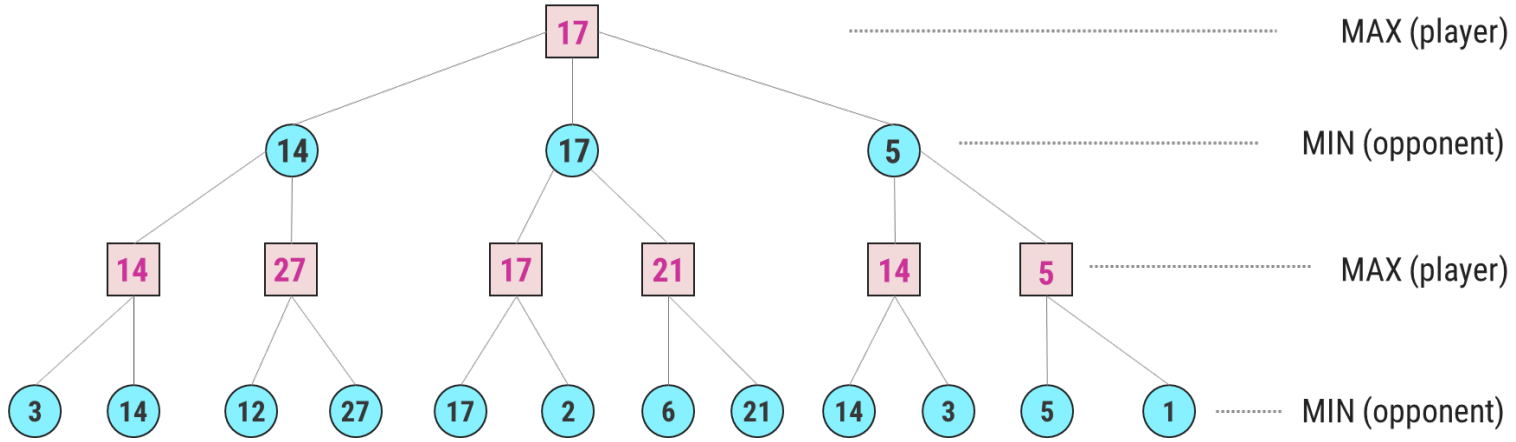
The MINIMAX Search Procedure

- The alternation of maximizing and minimizing at alternate ply when evaluations are to be pushed back up corresponds to the opposing strategies of the two players is called MINIMAX.

- It is recursive procedure that depends on two procedures :
  1. MOVEGEN(position, player)— The plausible-move generator, which returns a list of nodes representing the moves that can be made by Player in Position.
  2. STATIC(position, player) -- static evaluation function, which returns a number representing the goodness of Position from the standpoint of Player.
- To decide when recursive procedure should stop, variety of factors may influence the decision such as,
  - Has one side won?
  - How many ply have we already explored? Or how much time is left?
  - How stable is the configuration?

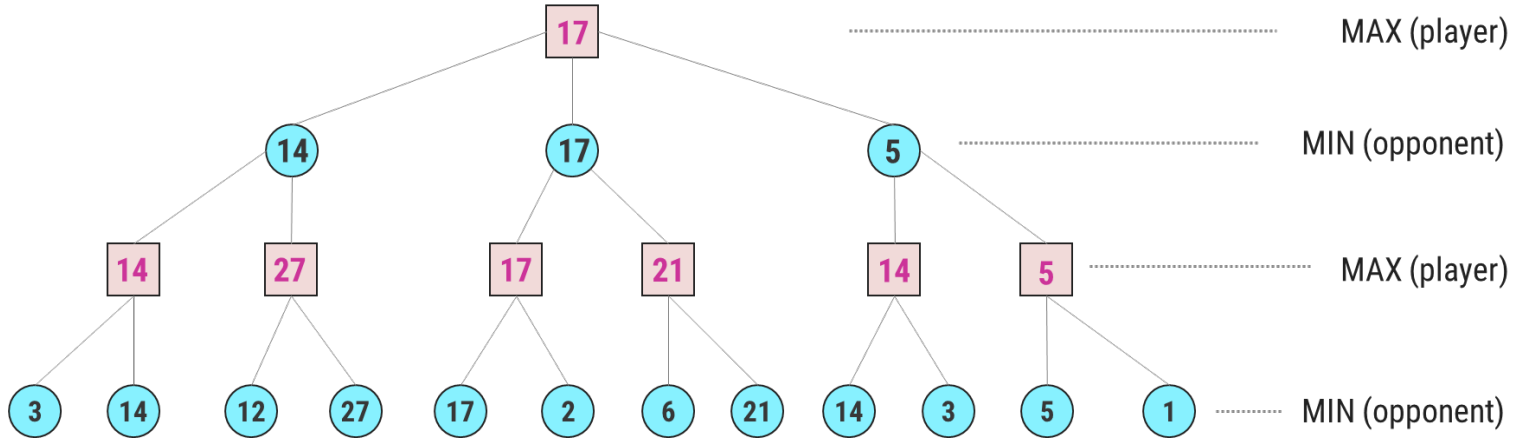
Minimax – Algorithm

- Given a game tree, the optimal strategy can be determined from the minimax value of each node, which we write as MINIMAX(n).
- The minimax algorithm computes the minimax decision from the current state.
- It uses a simple recursive computation of the minimax values of each successor state, directly implementing the defining equations.
- The recursion proceeds all the way down to the leaves of the tree, and then the minimax values are backed up through the tree as the recursion unwinds.



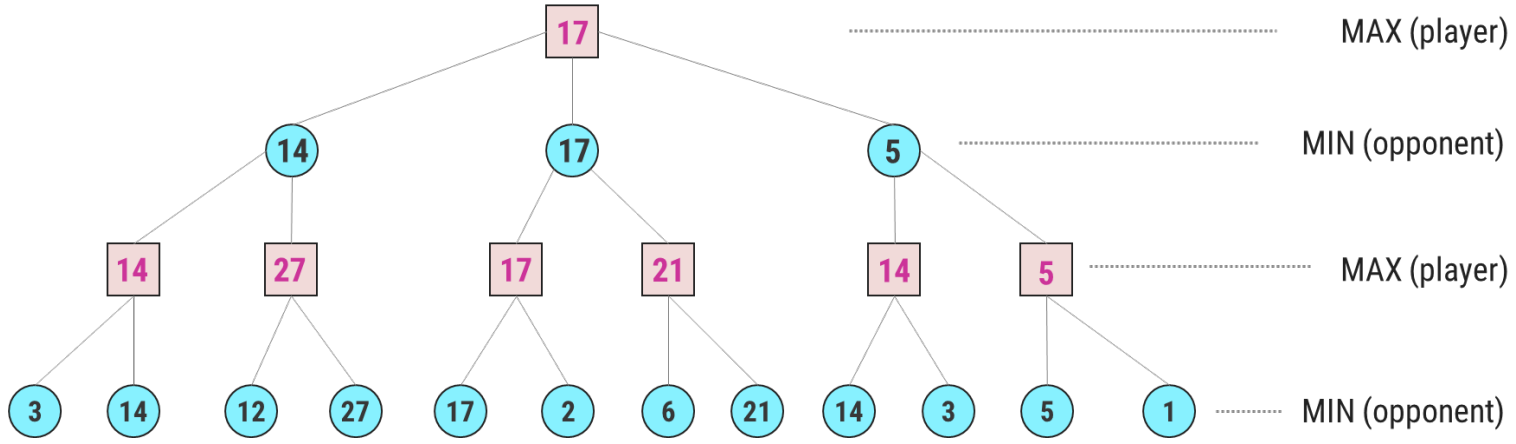
Minimax – Algorithm

- The minimax value of a node is the utility (for MAX) of being in the corresponding state, assuming that both players play optimally from there to the end of the game.
- The algorithm first recuses down to the three bottom left nodes and uses the UTILITY function on them to discover that their values are 3 and 14 respectively.
- Then it takes the maximum of these values, 14, and returns it as the backed up value of parent node.
- A similar process gives the backed-up values of 27, 17, 21, 14 and 5 respectively.



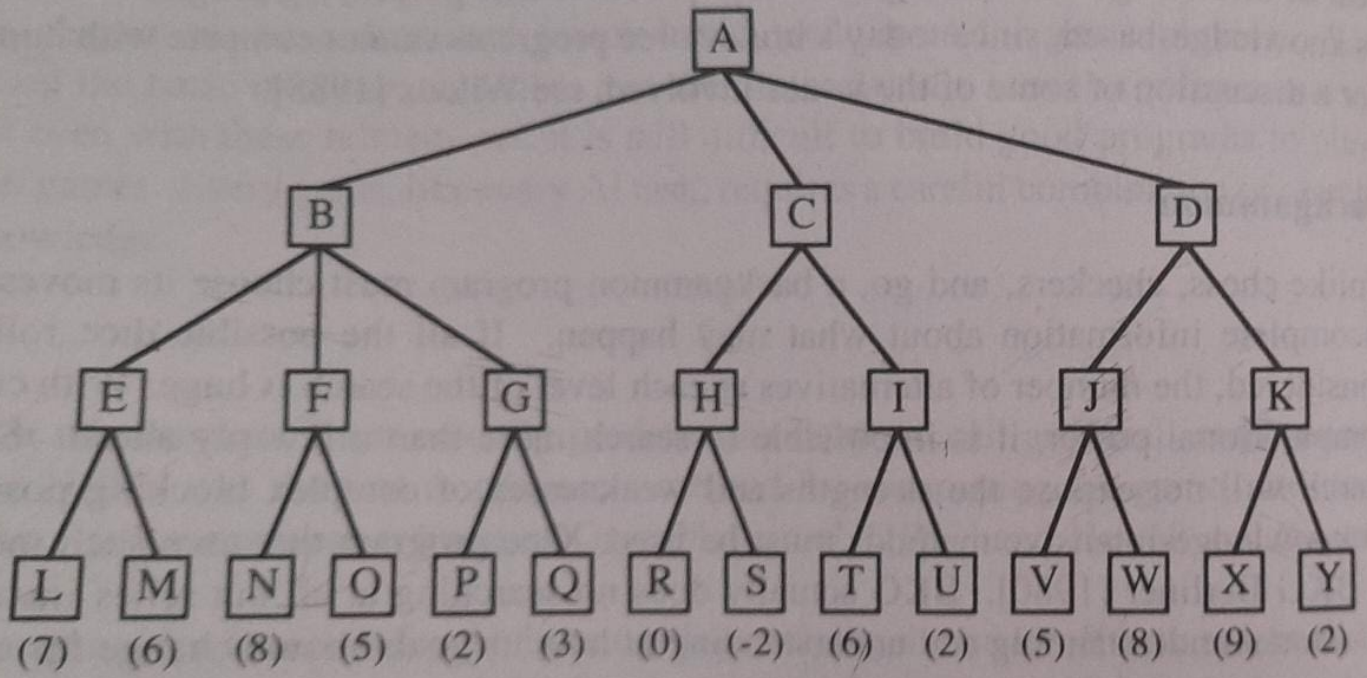
Minimax – Algorithm

- Then it takes the minimum of these values 14, 17 and 5 respectively.
- Finally, we take the maximum of 14, 17, and 5 to get the backed-up value of 17 for the root node.
- The minimax algorithm performs a complete **depth-first exploration** of the game tree.
- During every ply MAX prefers to move to a state of maximum value, whereas MIN prefers a state of minimum value.



Exercises

1. Consider the following game tree in which static scores are all from the first player's point of view:



Suppose the first player is the maximizing player. What move should be chosen?

Alpha-Beta Pruning

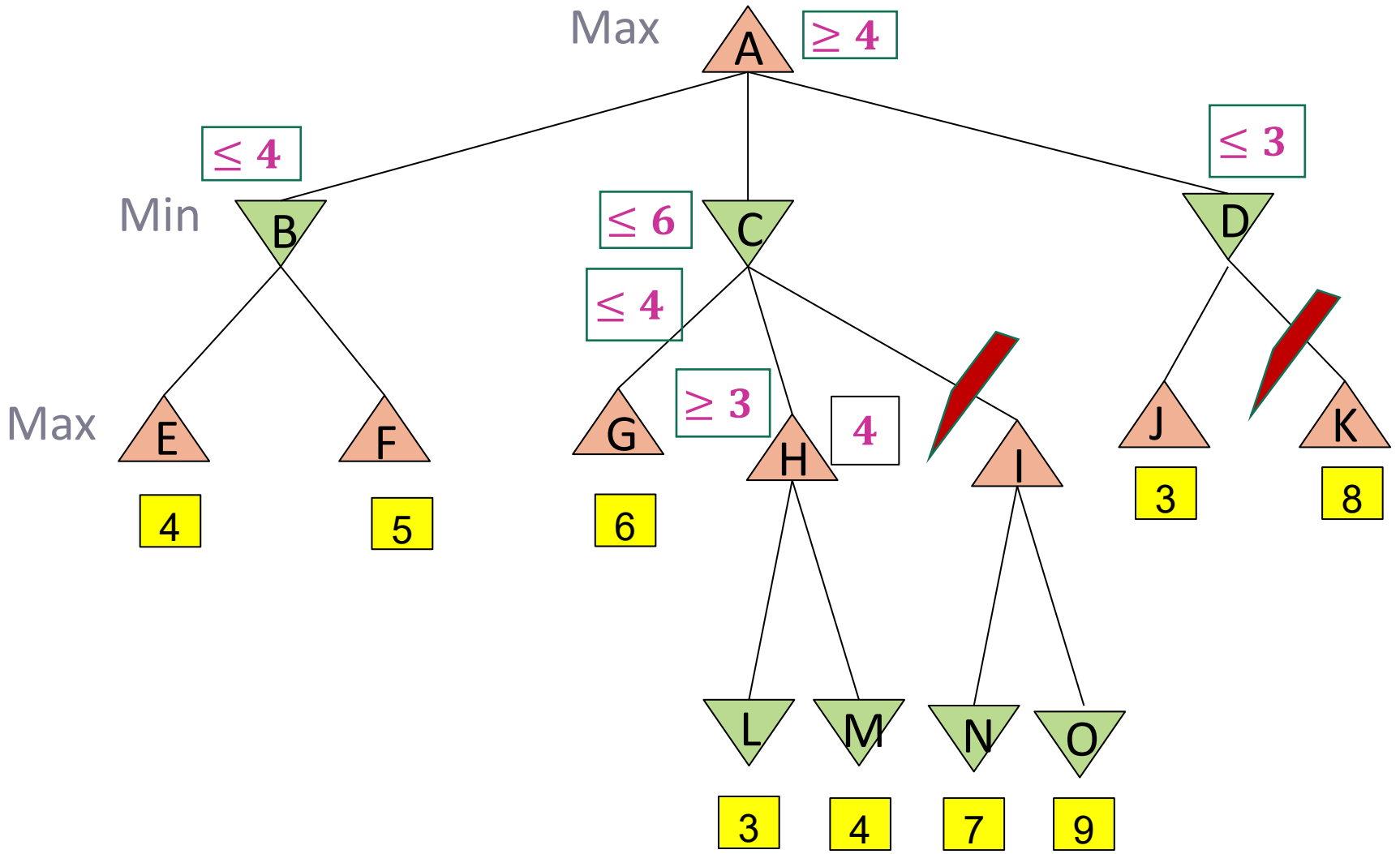
- Alpha-beta pruning is a modified version of the Minimax algorithm. It is an optimization technique for the Minimax algorithm.
- In the Minimax search algorithm, the number of game states to be examined can be exponential in the depth of a tree.
- Hence there is a technique by which without checking each node of the game tree we can compute the correct Minimax decision, and this technique is called pruning.
- This involves two threshold parameter Alpha and beta for future expansion, so it is called alpha-beta pruning. It is also called as Alpha-Beta Algorithm.
- Alpha-beta pruning can be applied at any depth of a tree, and sometimes not only it prunes the tree leaves but also entire sub-tree.

Alpha-Beta Pruning

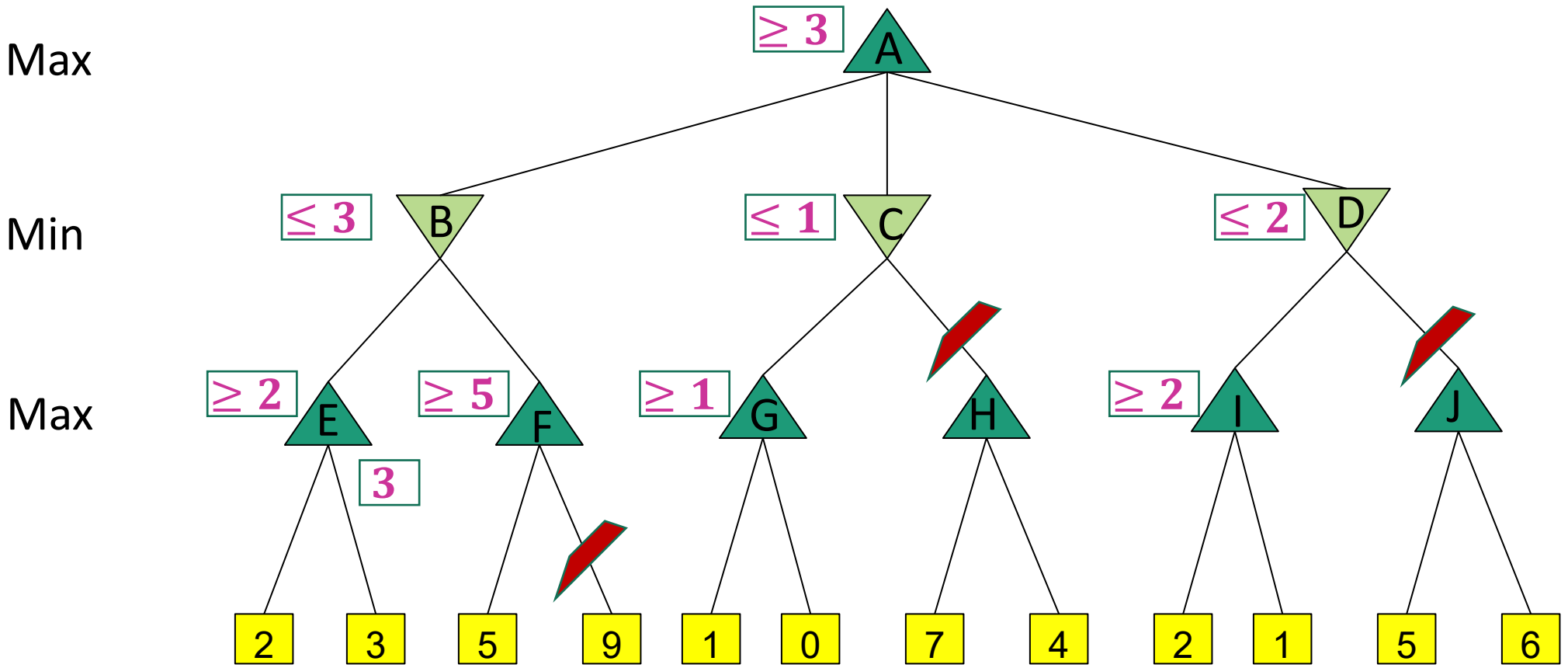
- Alpha-beta pruning technique maintains two bounds:
  1. Alpha (α): The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is -∞. A lower bound on best, i.e., Max
  2. Beta (β): The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is +∞. An upper bound on what the opponent can achieve

- The Search proceeds maintaining α and β
- Whenever $\alpha \geq \beta\_higher$, or $\beta \leq \alpha\_higher$, searching further at this node is irrelevant.
- In conclusion, Minimax with alpha beta pruning is a faster algorithm than the Minimax algorithm.

Minimax with Alpha-beta Pruning – Example 1



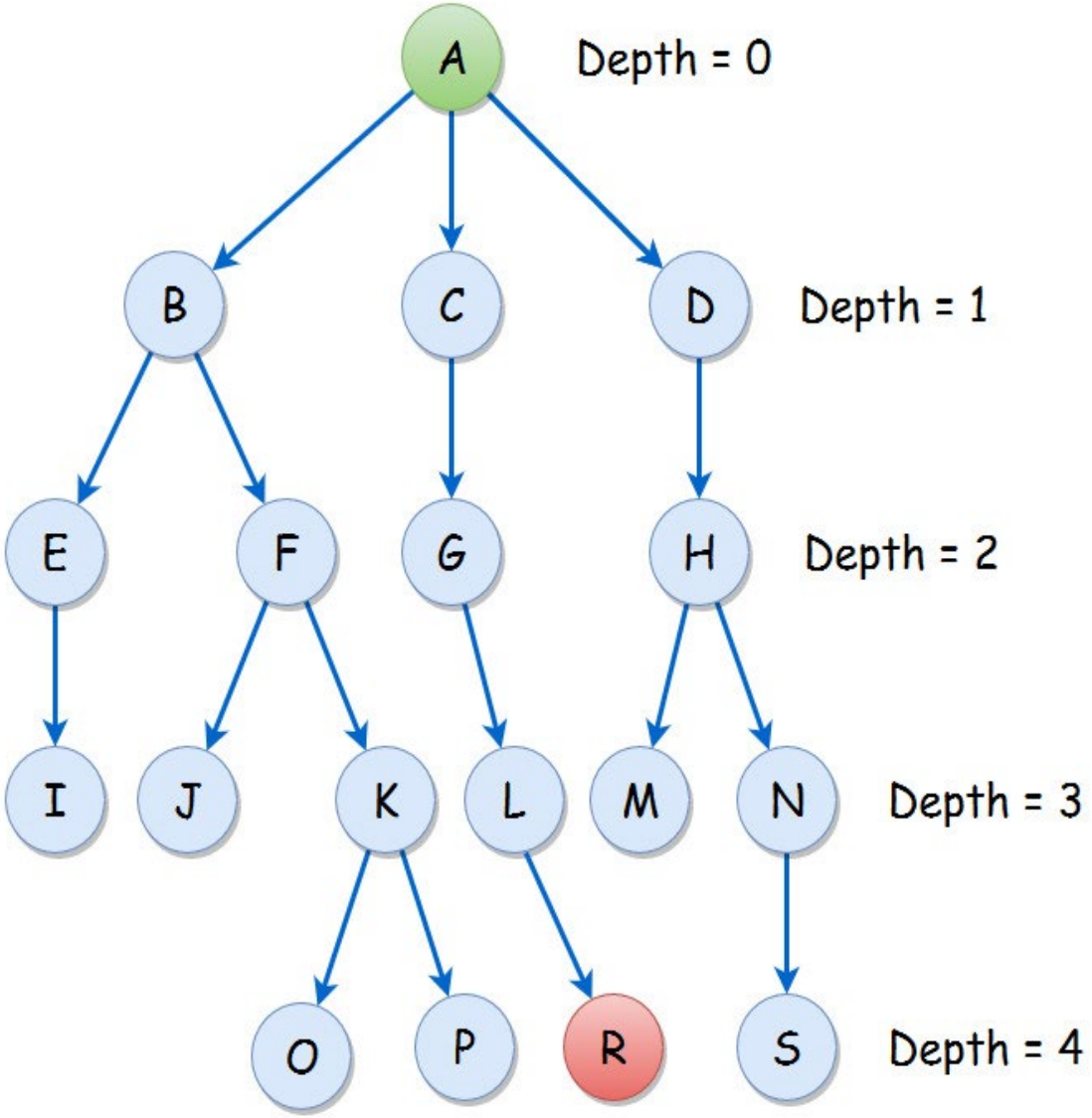
Minimax with Alpha-beta Pruning – Example 2



Iterative Deepening Search

- Depth first search is incomplete if there is an infinite branch in the search tree.
- Infinite branches can happen if:
  - paths contain loops
  - infinite number of states and/or operators.
- For problems with infinite state spaces, several variants of depth-first search have been developed: depth limited search, iterative deepening search.
- Depth limited search expands the search tree depth-first up to a maximum depth l.
- The nodes at depth $\boldsymbol{l}$ are treated as if they had no successors.
- Iterative deepening (depth-first) search (IDDS) is a form of depth limited search which progressively increases the bound.
- It first tries $l = 1$, then $l = 2$, then $l = 3$, etc. until a solution is found at $l = d$.

Iterative Deepening Search



Depth	DLS Traversal
0	A
1	A B C D
2	A B E F C G D H
3	A B E I F J K C G L D H M N
4	A B E I F J K O P C G L R

Thank You!