CS249 – ARTIFICIAL INTELLIGENCE - 1

7th Week Slides by:

2201AI33 – Saksham Singh
2201AI34 – Samar Kumar Srivastava
2201AI35 – Saumya Pratap Singh
2201AI36 – Somil Aggarwal
2201AI37 – Sowmya Raj

Problem Reduction Search

Introduction

Problem reduction search is a problem-solving strategy in artificial intelligence that involves breaking down a complex problem into smaller, more manageable subproblems until a solution is found. This approach is often used in domains where the problem space is large and complex, making traditional search methods inefficient. This strategy can be used in

1. Matrix Multiplication
2. Tower of Hanoi
3. Block word problem
4. Theorem Proving

Process of Problem Reduction Search:

- **Initial Problem**: The process begins with defining the initial problem, which may be complex and difficult to solve directly.
- **Identify Subproblems**: Break down the initial problem into smaller, more manageable subproblems. These subproblems should be easier to solve individually compared to the original problem.
- **Solve Subproblems**: Solve each subproblem using appropriate problem-solving techniques. This may involve using various algorithms, heuristics, or domain-specific knowledge.

- **Combine Solutions**: Once solutions to the subproblems are obtained, combine them to form a solution to the original problem. This step may involve integrating the solutions in a meaningful way or performing additional processing.
- **Verify Solution**: Finally, verify the solution to ensure that it satisfies all the constraints and requirements of the original problem. If the solution is not satisfactory, the process may need to backtrack and explore alternative approaches.

Example to illustrate the above theory

Consider the problem of finding the shortest path between two points in a maze. Instead of searching the entire maze at once, problem reduction search breaks it down into smaller subproblems:

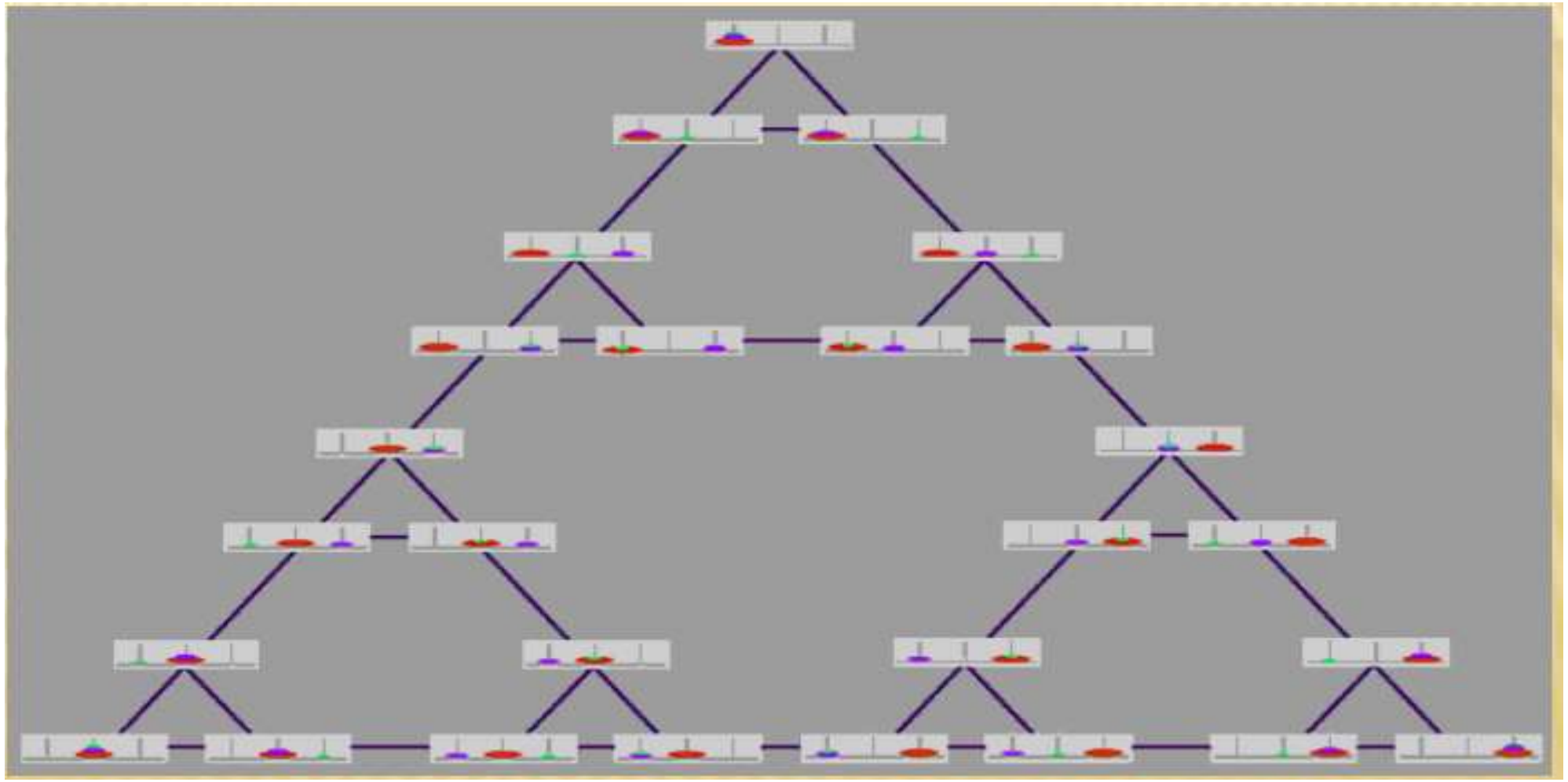
Solution:

- Subproblem 1: Determine the possible moves from the current position.
- Subproblem 2: Evaluate the cost of each possible move (e.g., distance to the destination).
- Subproblem 3: Select the move with the lowest cost.
- Subproblem 4: Repeat steps 1-3 until the destination is reached

Problem Reduction Search on Tower of Hanoi

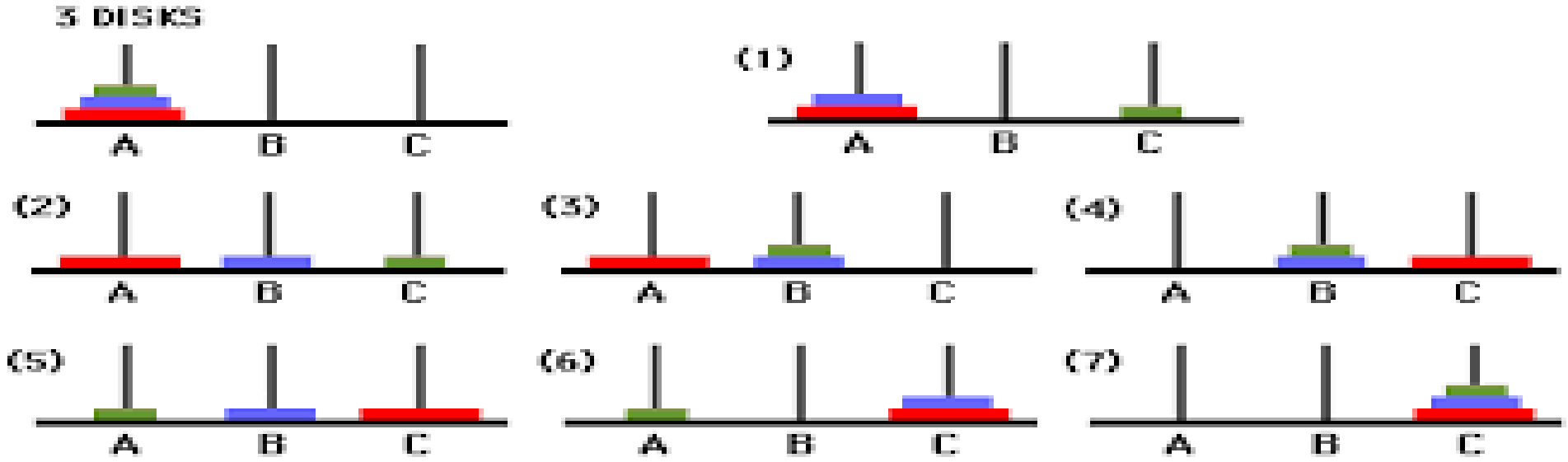
- The figure below shows the state space associated with a 3-disk Tower of Hanoi Problem. The problem involves moving from a state where the disks are stacked on one of the pegs and moving them so that they end up stacked on a different peg.
- In this case, we will consider the state at the top of the figure the starting state. In this case all three disks are on the left-most peg.
- And we will consider the state at the bottom right to be the goal state. In this state the three disks are now all stacked on the right-most peg.

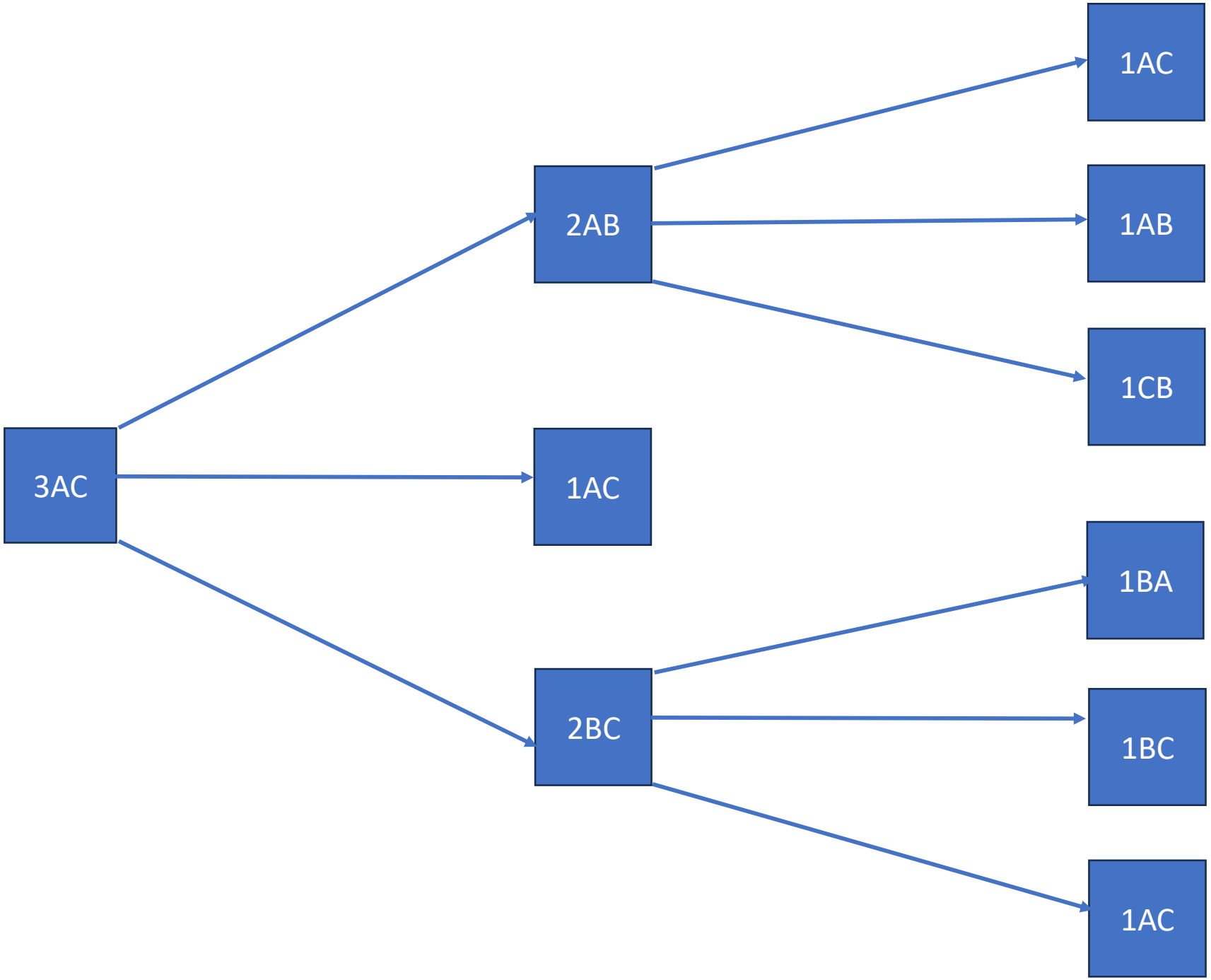
State Space for 3 Disk Tower of Hanoi



Problem Reduction Space for Tower of Hanoi

- The root node, labelled “3AC” represents the original problem of transferring all 3 disks from peg A to peg C.
- The goal can be decomposed into three subgoals: 2AB, 1AC, 2BC. In order to achieve the goal, all 3 subgoals must be achieved.




3AC
2AB
1AC
2BC
1AC
1AB
1CB
1BA
1BC
1AC

Advantages of Problem Reduction Search:

1. Efficiency: By breaking down the problem into smaller parts, problem reduction search can often find solutions more efficiently than exhaustive search methods.
2. Modularity: The approach lends itself well to modular problem-solving, where different subproblems can be tackled independently.
3. Flexibility: Problem reduction search can be adapted to various problem domains and can incorporate different problem-solving techniques as needed.

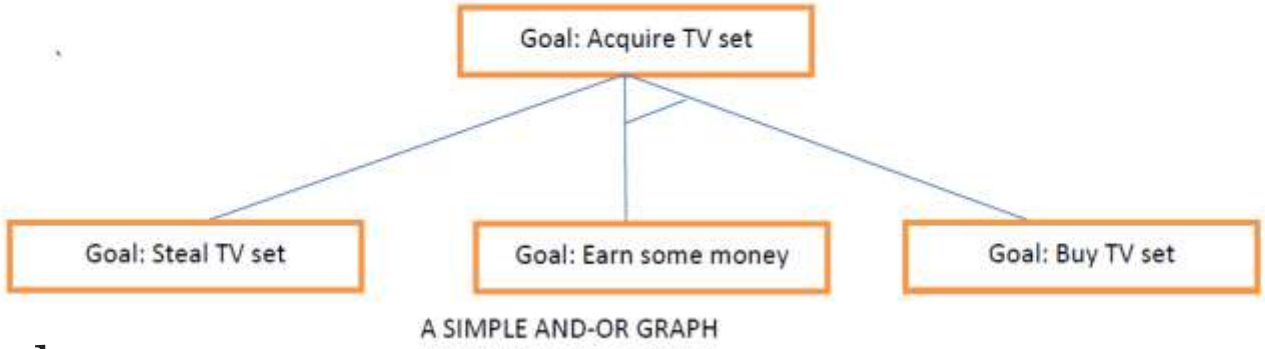
Limitations:

1. Complexity: The process of identifying and solving subproblems may introduce additional complexity, particularly in highly interconnected problem spaces.
2. Search Space: Depending on the structure of the problem, problem reduction search may still involve exploring a large search space, leading to computational challenges.
3. Optimality: The solution obtained through problem reduction search may not always be optimal, especially if the decomposition of the problem leads to overlooking certain aspects.

AND/OR Graph

The AND-OR GRAPH (or tree) is useful for representing the solution of problems that can 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 number of successor nodes, all of which must be solved in order for the arc to point to a solution. Just as in an OR graph, several arcs may emerge from a single node, indicating a variety of ways in which the original problem might be solved. This is why the structure is called not simply an AND-graph but rather an AND-OR graph (which also happens to be an AND-OR tree).

The AND/OR graph search problem



A SIMPLE AND-OR GRAPH

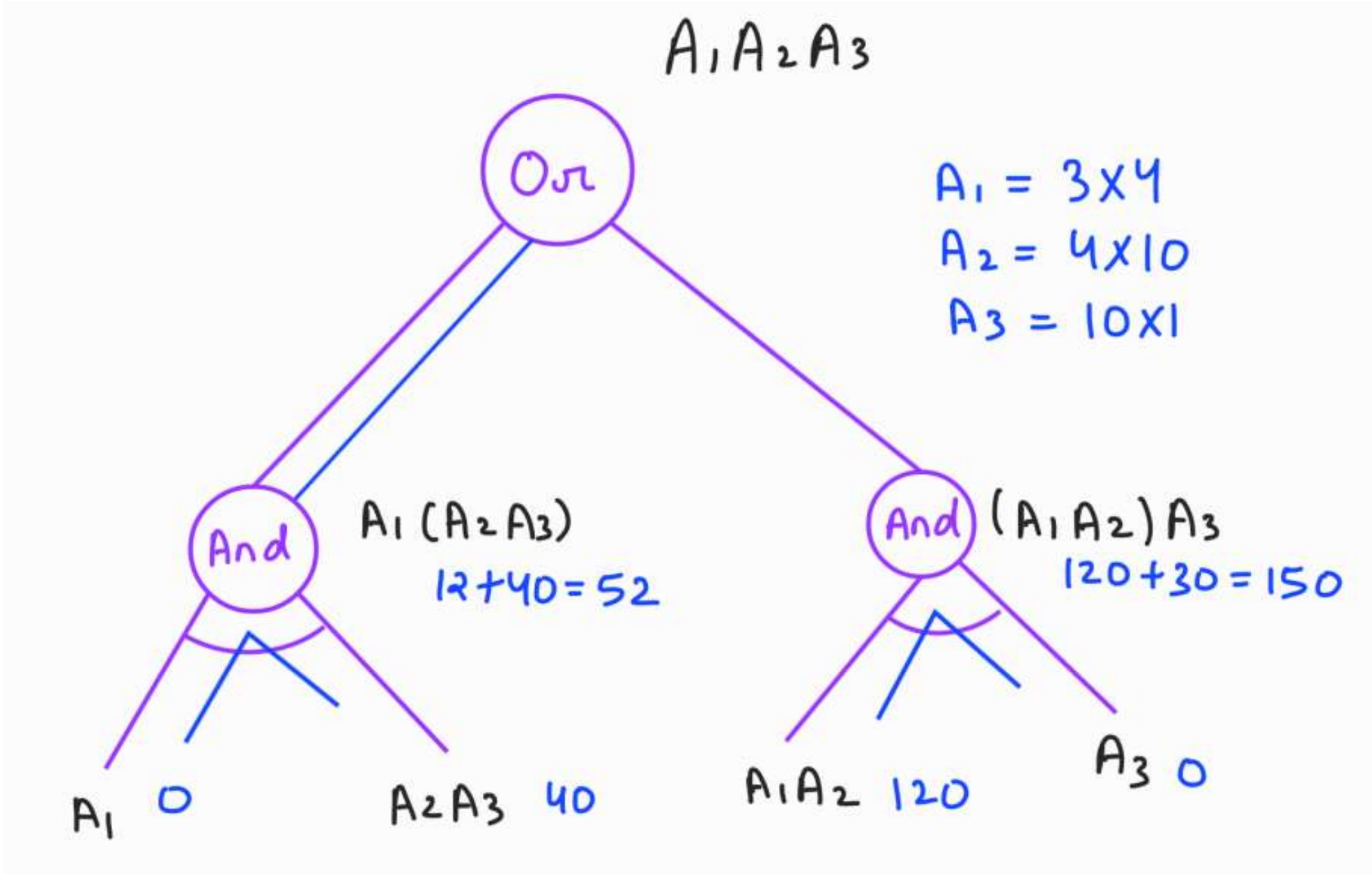
Problem definition : -

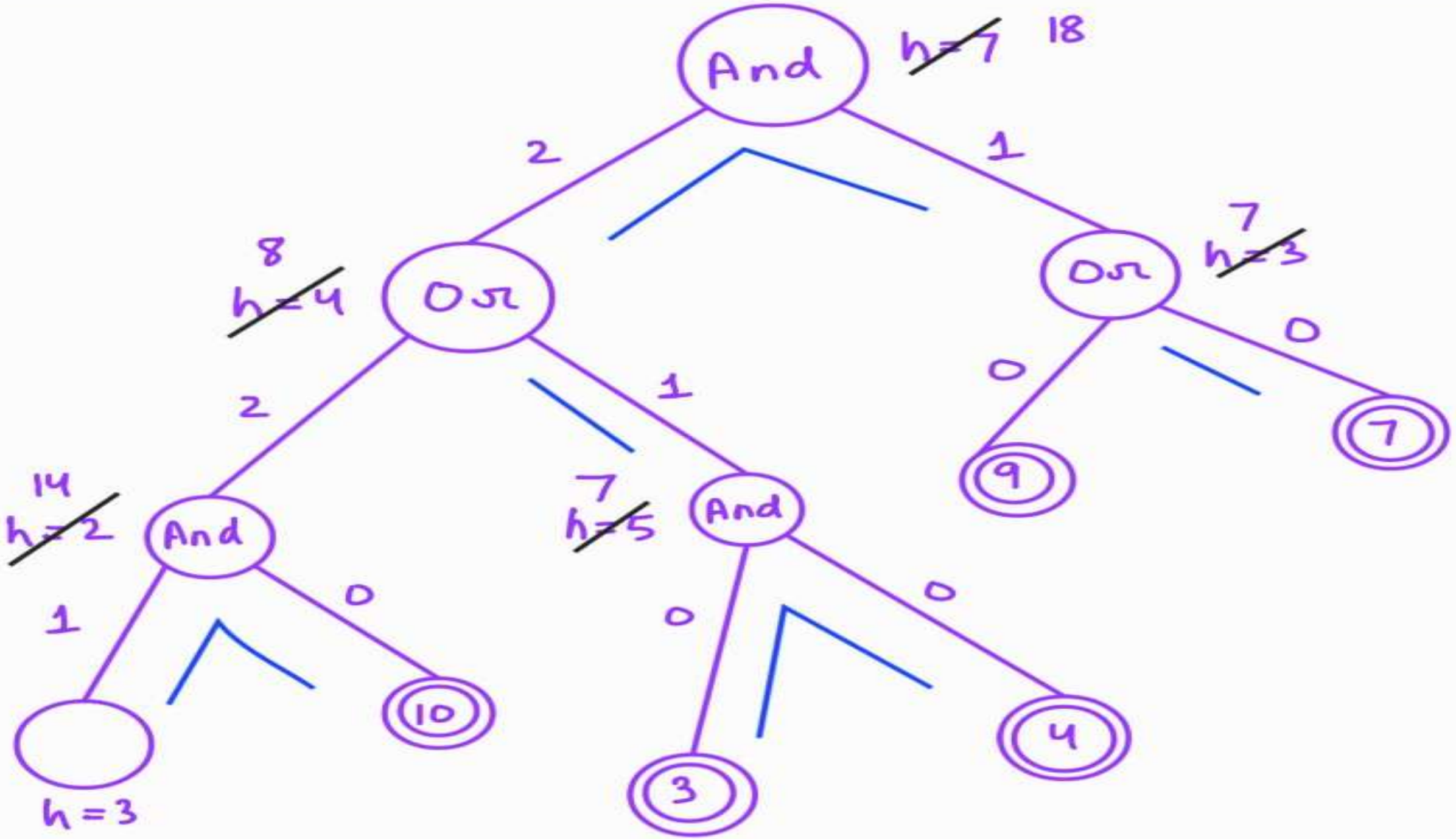
Given : [G, s, T] where

- G: implicitly specified AND/OR graph
- S: start node of the AND/OR graph
- T: set of terminal nodes
- h(n) heuristic function estimating the cost of solving the sub -problem at n –

To find : –

A minimum cost solution tree for matrix multiplication of below given probem


A1A2A3
Or
A1 = 3x4
A2 = 4x10
A3 = 10x1
And
A1 (A2A3)
12+40=52
And
(A1A2)A3
120+30=150
A1 0
A2A3 40
A1A2 120
A3 0


And
h=7
18
2
1
8
h=4
Or
Or
7
h=3
2
1
0
0
7
14
h=2
7
h=5
And
And
9
7
1
0
0
0
10
4
3
h=3

AO*Algorithm

Introduction

- It is an informed search and works as best first search.
- AO* Algorithm is based on problem decomposition (Breakdown problem into small pieces).
- It is an efficient method to explore a solution path.
- AO* is often used for the common pathfinding problem in applications such as video games, but was originally designed as a general graph traversal algorithm.
- It finds applications in diverse problems, including the problem of parsing using stochastic grammars in NLP.
- Other cases include an Informational search with online learning.
- It is useful for searching game trees, problem solving etc.

Algorithm

1>> Initialize: Set G* = {s}, f(s) = h(s)

if s ∈ T, label s as SOLVED

2>> Terminate: If s is SOLVED, then Terminate.

3>> Select: Select a non-Select a non-terminal leaf node n from the marked

sub-tree/sub-graph.

4>> Expand: Make explicit the successors of n, for each new successor m:

Set f(m) = h(m)

if m is TERMINAL, label m SOLVED

5>> Cost Revision: Call cost-revise(n)

6>> Loop: Go To Step 2

Cost Revision in AO*

1>> Create Z = {n}

2>> If Z = { }, Return

3>> Select a node m from Z such that m has no descendants in Z

4>> If m is an AND node with successors r_1, r_2, … r_k

Set f(m) =Σ [f(r_i) + c(m, r_i)]

Mark the edge to each successor of m if each successor is labeled as SOLVED

5>> If m is an OR node with successors r_1, r_2, … r_k

Set f(m) = min f(m) = min [f(r_i) + c(m, r_i)]

Mark the edge to the best successor of m if the marked successor is labeled as SOLVED then label m as SOLVED.

6>> If the cost or label of m has changed, then insert THOSE PARENTS of m into Z for which m is a marked successor.

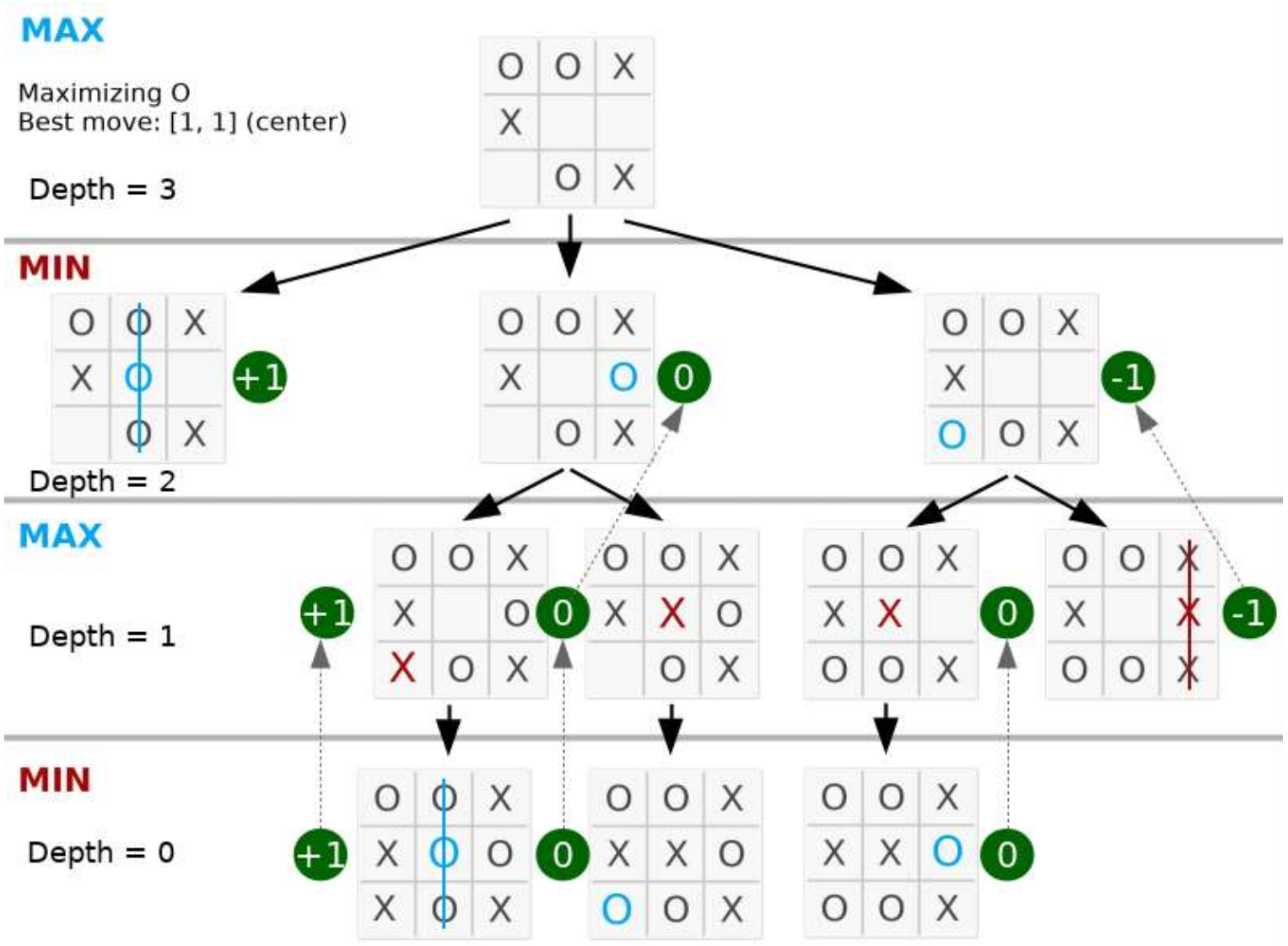
7>> Go to Step 2.

If there are only OR nodes in the graph then AO* is synonymous to A*

Searching Game Trees

- Consider an OR Tree with 2 types of OR nodes i.e MIN Nodes.
- In MIN nodes select MIN Cost Successors.
- In MAX nodes select MAX Cost Successors.
- TERMINAL nodes are winning/loosing states;
  - it is infeasible to search up to the term nodes.
  - We use Heuristic costs to compare non-terminal nodes.

Game Tree For Tic-Tac-Toe



Adversarial Search

Introduction

- Adversarial search problems = games
- They occur in multiagent competitive environments
- There is an opponent we can't control planning again us!
- Game vs. search: optimal solution is not a sequence of actions but a strategy (policy) If opponent does *a*, agent does *b*, else if opponent does *c*, agent does *d*, etc.
- Tedious and fragile if hard-coded (i.e., implemented with rules)
- Good news: Games are modeled as search problems and use heuristic evaluation functions

Types of Games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

Zero Sum Games

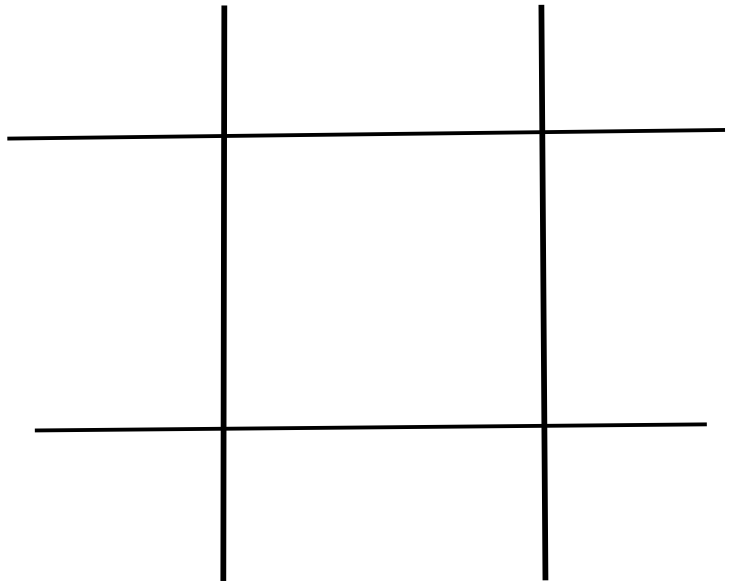
- Adversarial: Pure competition.
- Agents have different values on the outcomes.
- One agent maximizes one single value, while the other minimizes it.
- Each move by one of the players is called a “ply".

One function: one agents maximizes it and one minimizes it!

Single Player Tic-Tac-Toe

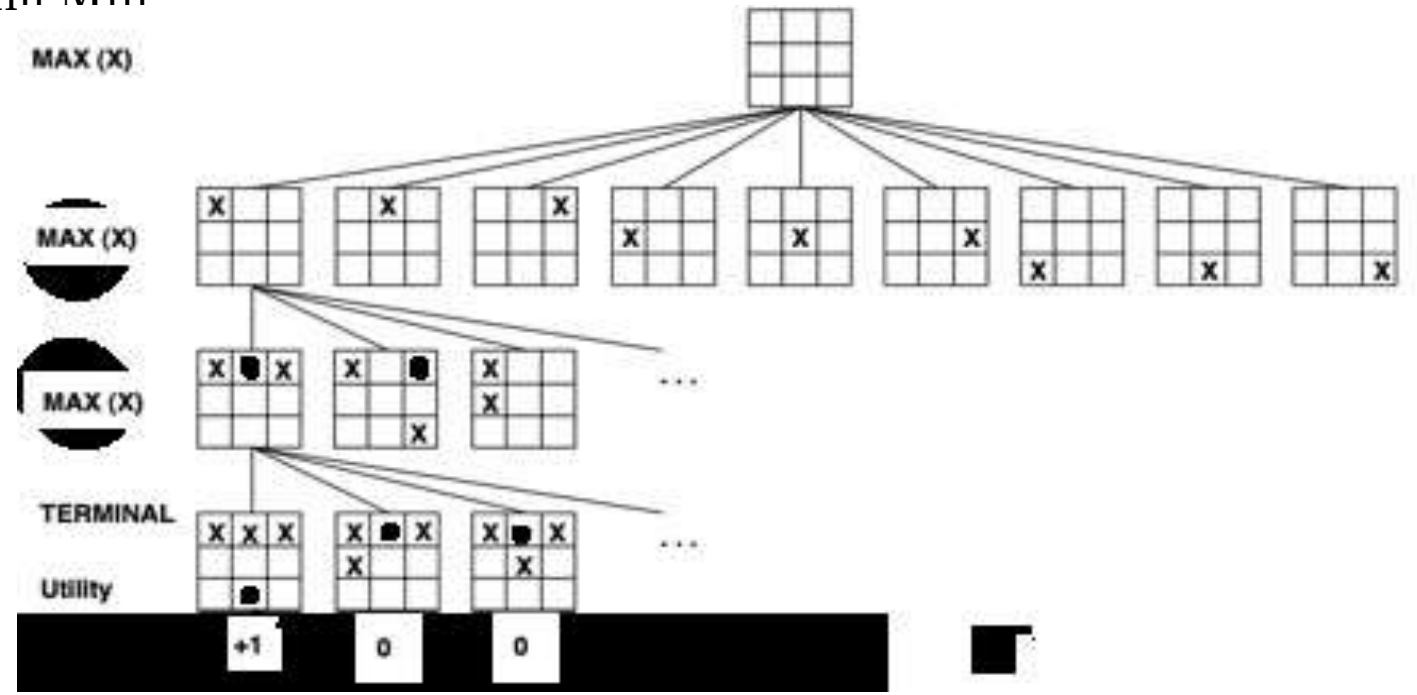
Assume we have a tic-tac-toe with one player.

Let’s call him Max and have him play three moves only for the sake of the example.



Continued.....

In the case of one player, nothing will prevent Max from winning (choose the path that leads to the desired utility here 1), unless there is another player who will do everything to make Max lose, let's call him Min



Adversarial search: minimax

- Search in Presence of Opponent
- Search Agent/Player involving in the Game

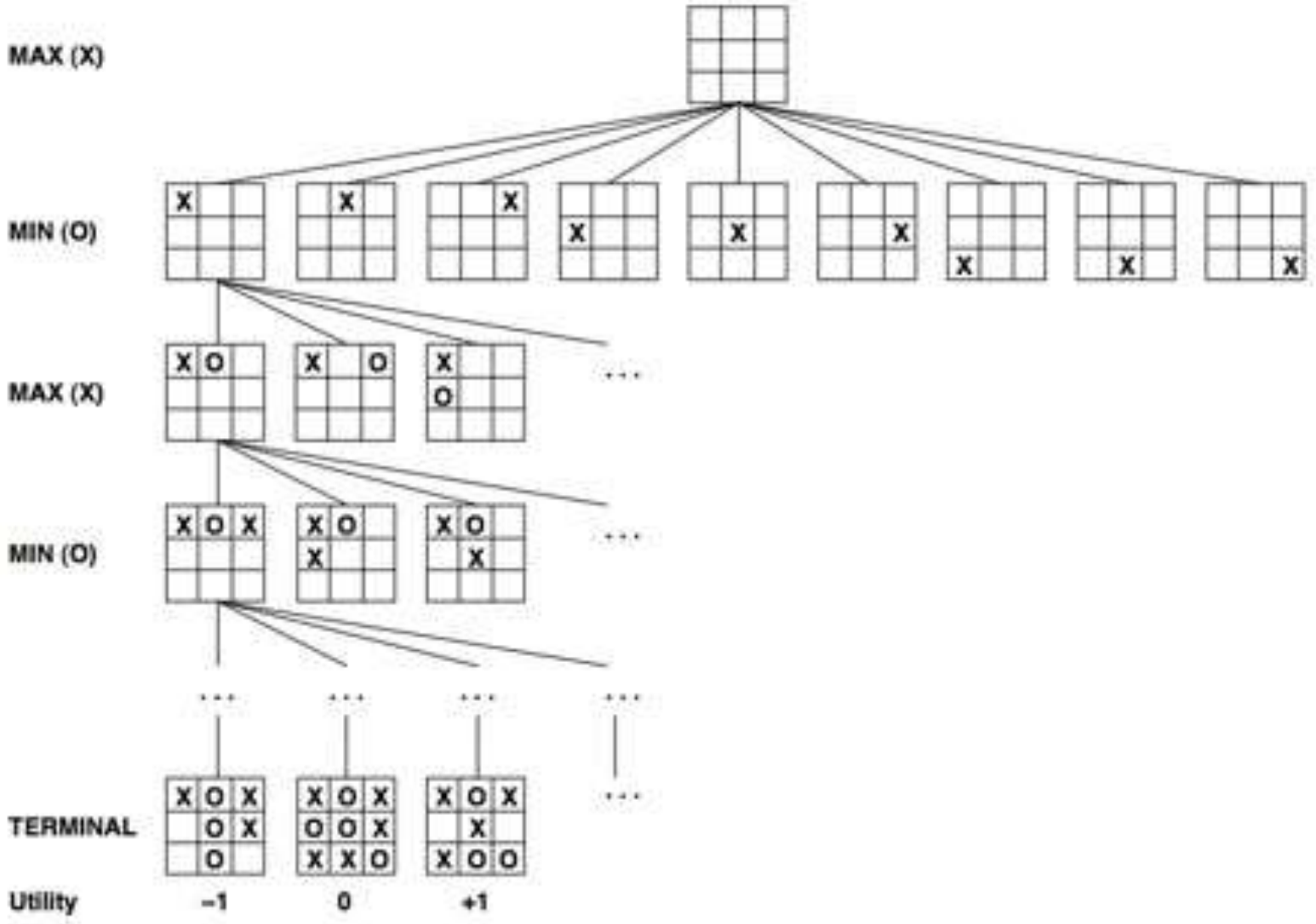
  Wants to MAXIMIZE its Payoff(i.e. Profit/Utility)
- The Sum of Payoffs may be constant(in case of zero sum game)

  OR may not be constant
- OPTIMIZING payoff in the presence of opponent

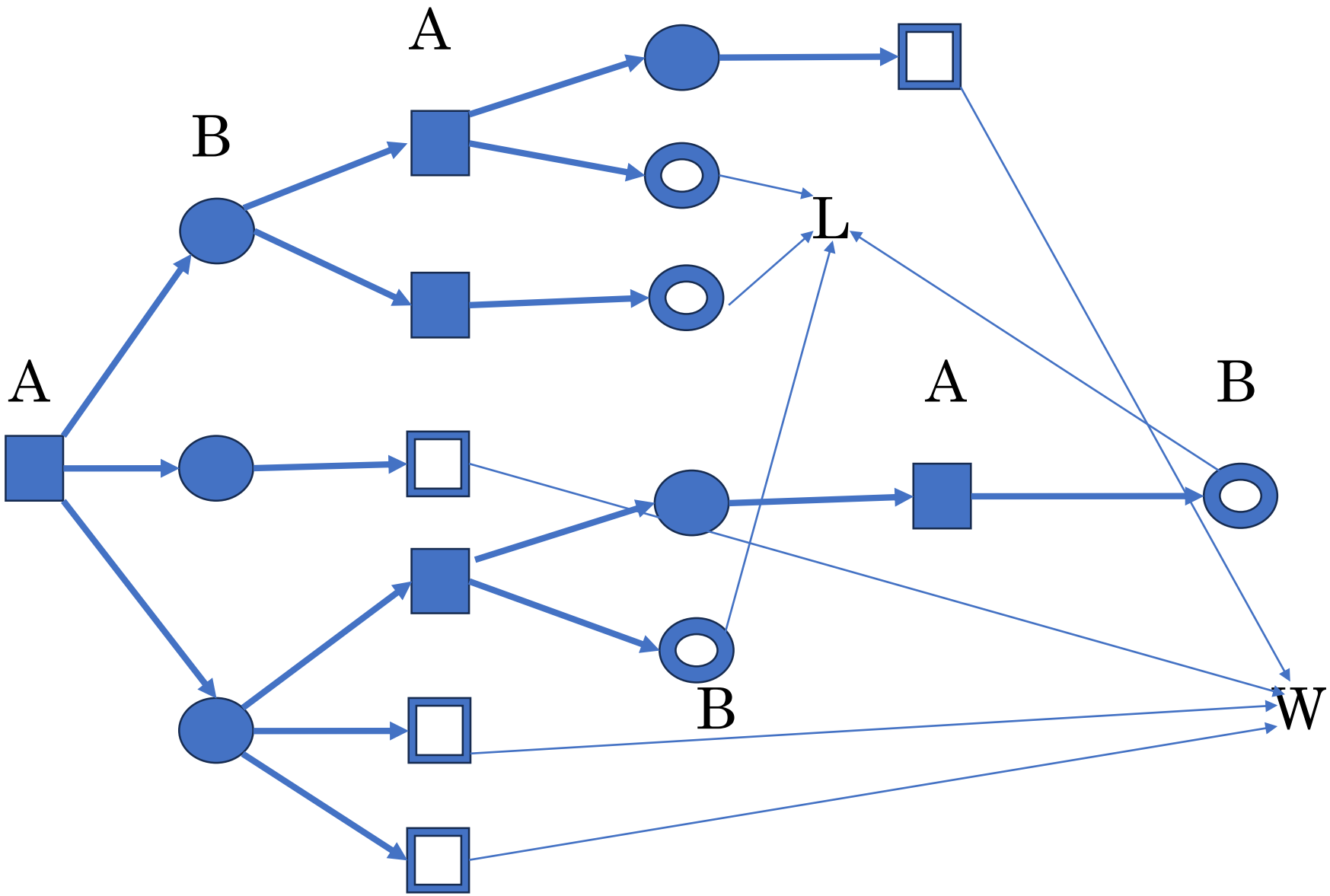
  - Profits are shared among the agents at each state of Game

  - Each agent aims to reach a state that maximizes its payoff in each turn of the Game.
- Example – Tic-Tac-Toe, CHESS

Minimax example



MINMAX Tree for Above Example



Game Theory

Prisoner's Dilemma

A CLASSICAL EXAMPLE OF GAME, a situation where individual decision makers always have an INCENTIVE to choose in a way that creates a less than optimal outcome for the individuals as a group

EXAMPLE -

You and your partner have both been caught red handed near the scene of a burglary(~~2 students have been caught for giving proxy~~).Both of you have been brought to the police station, where you are interrogated **separately** by the police.

Prisoner's Dilemma

The police present your options:

1. You can either CONFESS

OR

2. You can remain SILENT

Here are the consequences of your actions:

- If you CONFESS and your partner remains SILENT, you are released and your partner will serve 10 years in jail
- If you remain SILENT and your partner CONFESSES, you will serve 10 years in jail and your partner is released.
- If both of you CONFESS, both of you will serve 5 years in jail
- If both of you remain SILENT, both of you will only serve 1 year in jail

Prisoner's Dilemma

Normal Form Representation :

Players:

ALICE & BOB

	-BOB- CONFESS	- BOB- REMAINS SILENT
-ALICE- CONFESS	A = -5 B = -5	A = 0 B = -10
- ALICE- REMAINS SILENT	A = -10 B = 0	A = -1 B = -1

Payoff :

For Each Player

Actions :

CONFESS

Or

REMAIN SILENT

Strategies

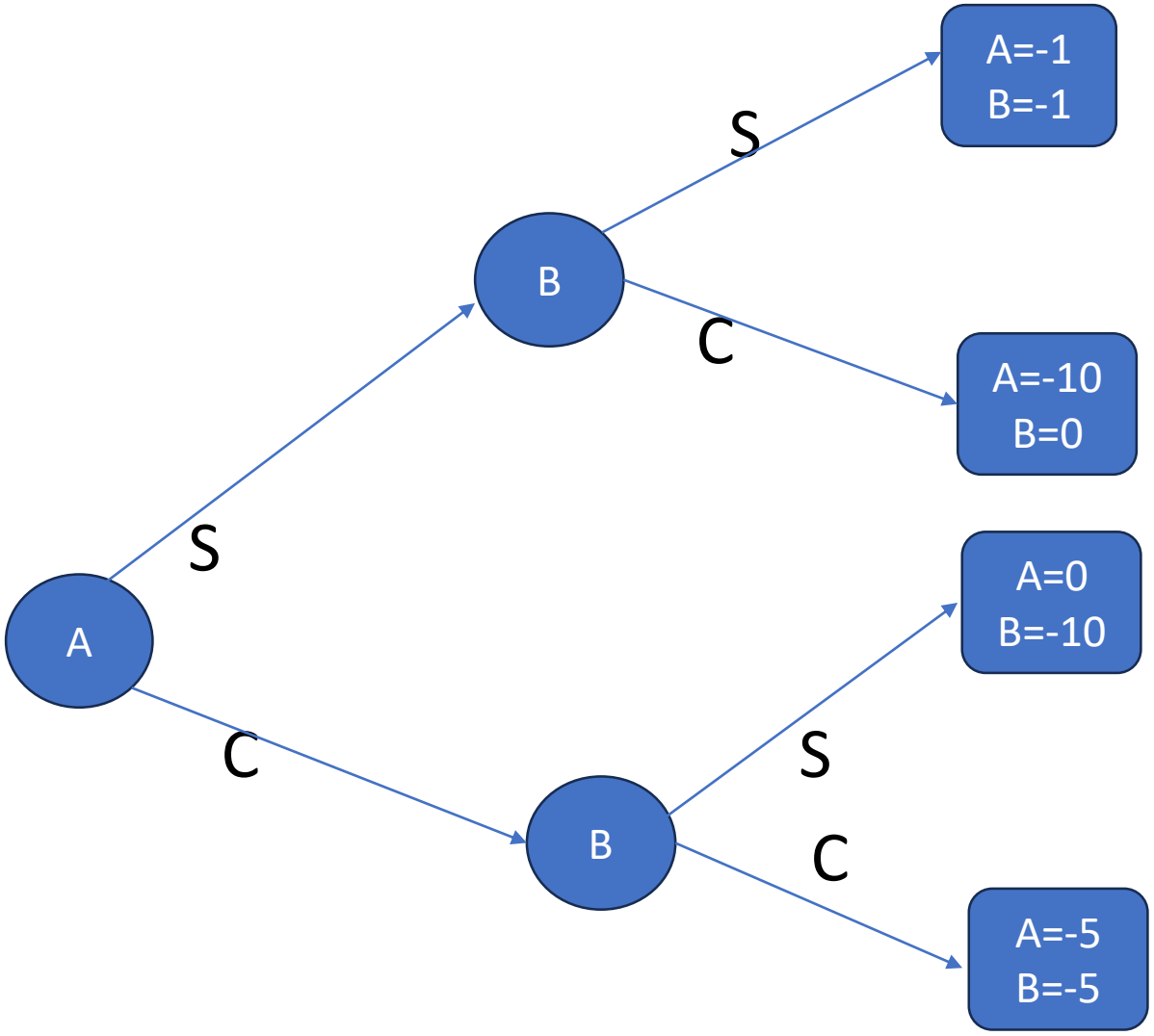
- Each player must adopt and execute a strategy
- Strategy = policy i.e. mapping from state to action
- Prisoner's Dilemma is a one move game:
  - Strategy is a single action
  - There is only a single state
- A pure strategy is a deterministic policy

Prisoner's Dilemma Strategy

- What is the right pure strategy for Alice or Bob?
- (Assume both want to maximize their own expected utility)

	-BOB- CONFESS	- BOB- REMAINS SILENT
-ALICE- CONFESS	A = -5 B = -5	A = 0 B = -10
- ALICE- REMAINS SILENT	A = -10 B = 0	A = -1 B = -1

Tree for above Example



Prisoner's Dilemma Strategy

Alice thinks:

- If Bob confesses, I get 5 years if
  I confess and 10 years if I don't
- If Bob doesn't confesses,
  I get 0 years if I confess and 1 year if I don't
- "Alright I'll testify"

	-BOB- CONFESS	- BOB- REMAINS SILENT
-ALICE- CONFESS	A = -5 B = -5	A = 0 B = -10
- ALICE- REMAINS SILENT	A = -10 B = 0	A = -1 B = -1

Prisoner's Dilemma Strategy

CONFESS is a **dominant strategy** for the game (notice how the payoffs for Alice are always bigger if she testifies than if she refuses)

	-BOB- CONFESS	- BOB- REMAINS SILENT
-ALICE- CONFESS	A = -5 B = -5	A = 0 B = -10
- ALICE- REMAINS SILENT	A = -10 B = 0	A = -1 B = -1

Dominant Strategies

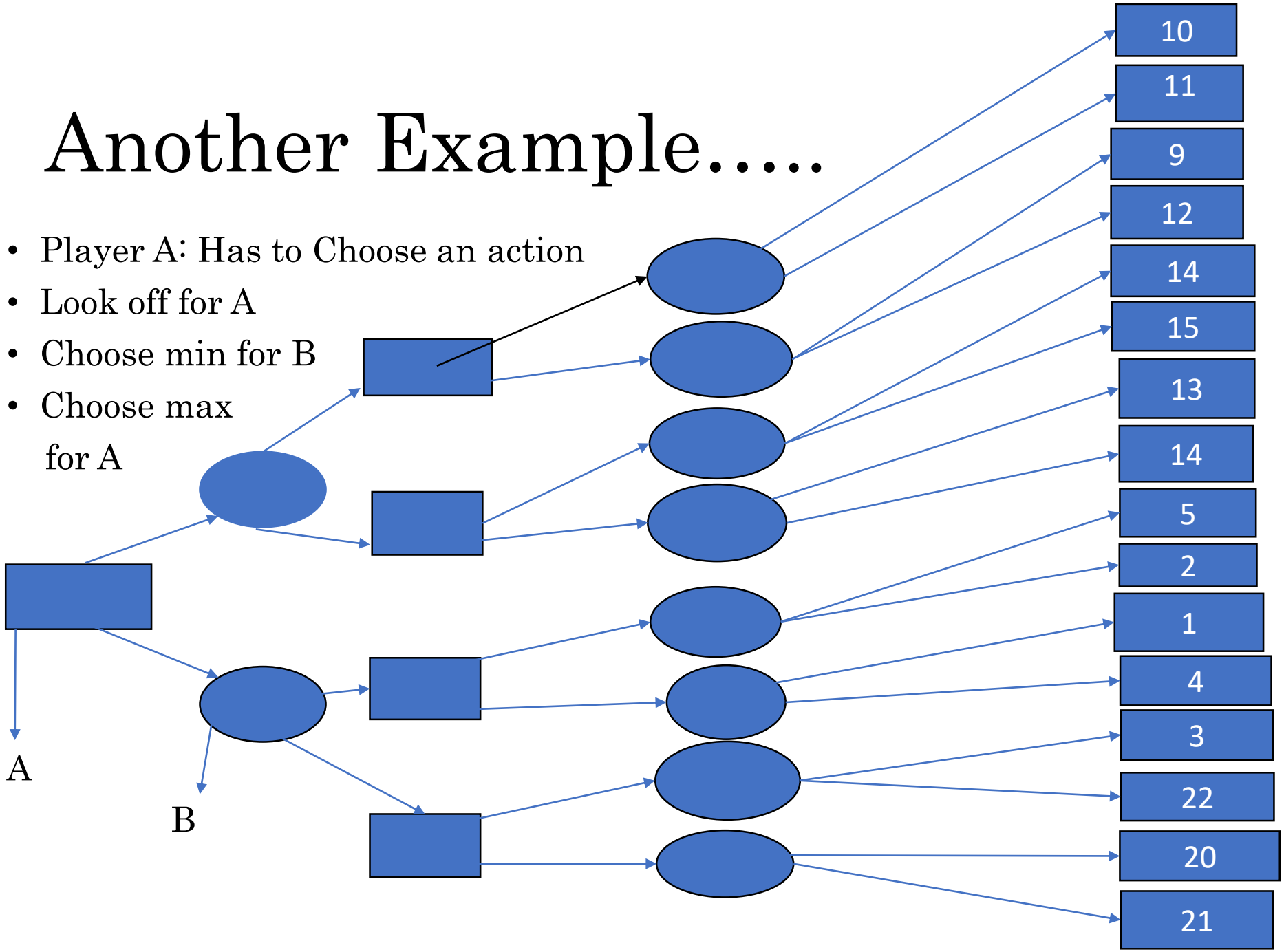
- Suppose a player has two strategies S and S'. We say S **dominates** S' if choosing S always yields at least as good an outcome as choosing S'.
- S **strictly dominates** S' if choosing S always gives a better outcome than choosing S' (no matter what the other player does)
- S **weakly dominates** S' if there is one set of opponent's actions for which S is superior, and all other sets of opponent's actions give S and S' the same payoff.

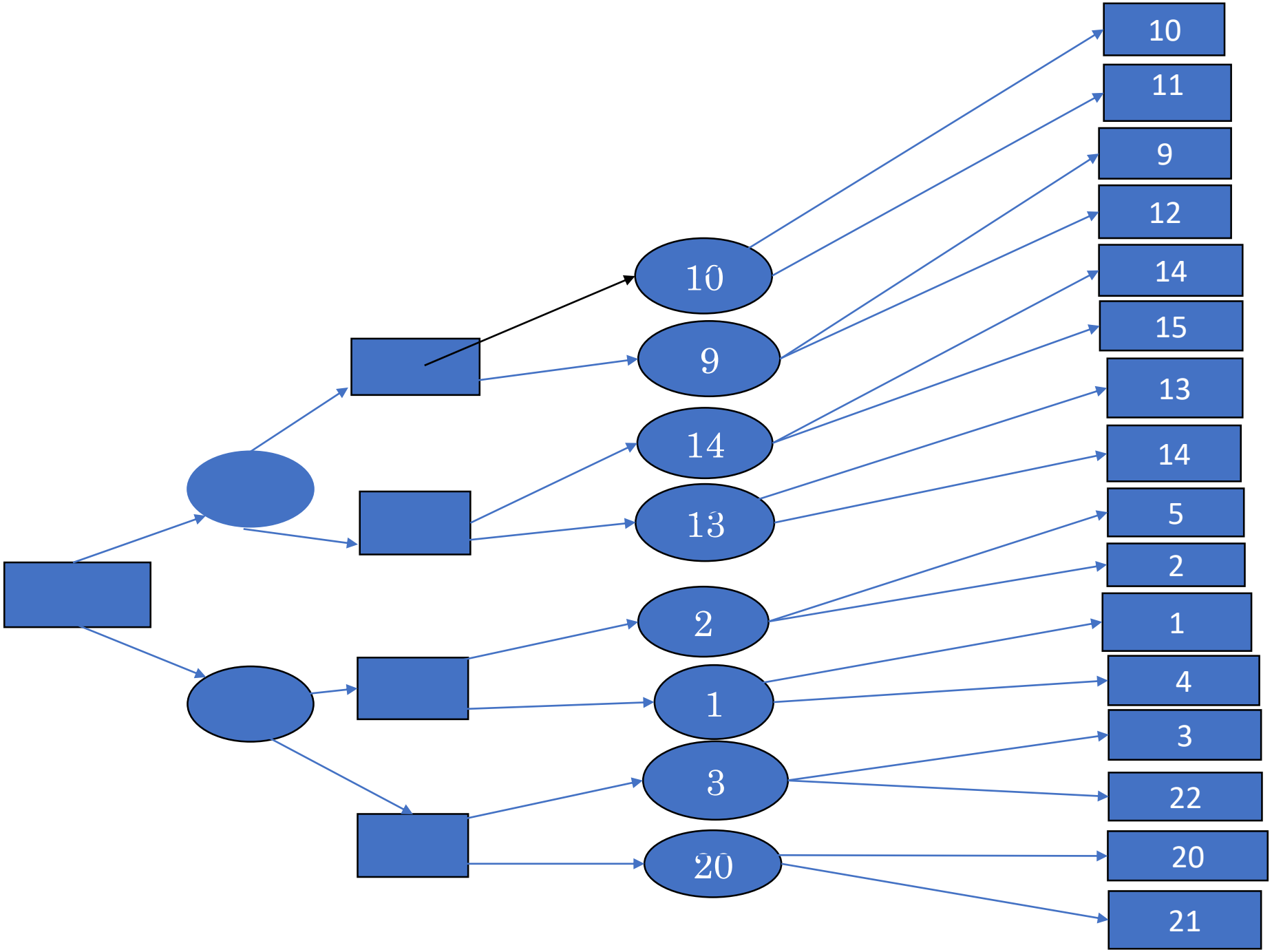
	-BOB- CONFESS	- BOB- REMAINS SILENT
-ALICE- CONFESS	A = -5 B = -5	A = 0 B = -10
- ALICE- REMAINS SILENT	A = -10 B = 0	A = -1 B = -1

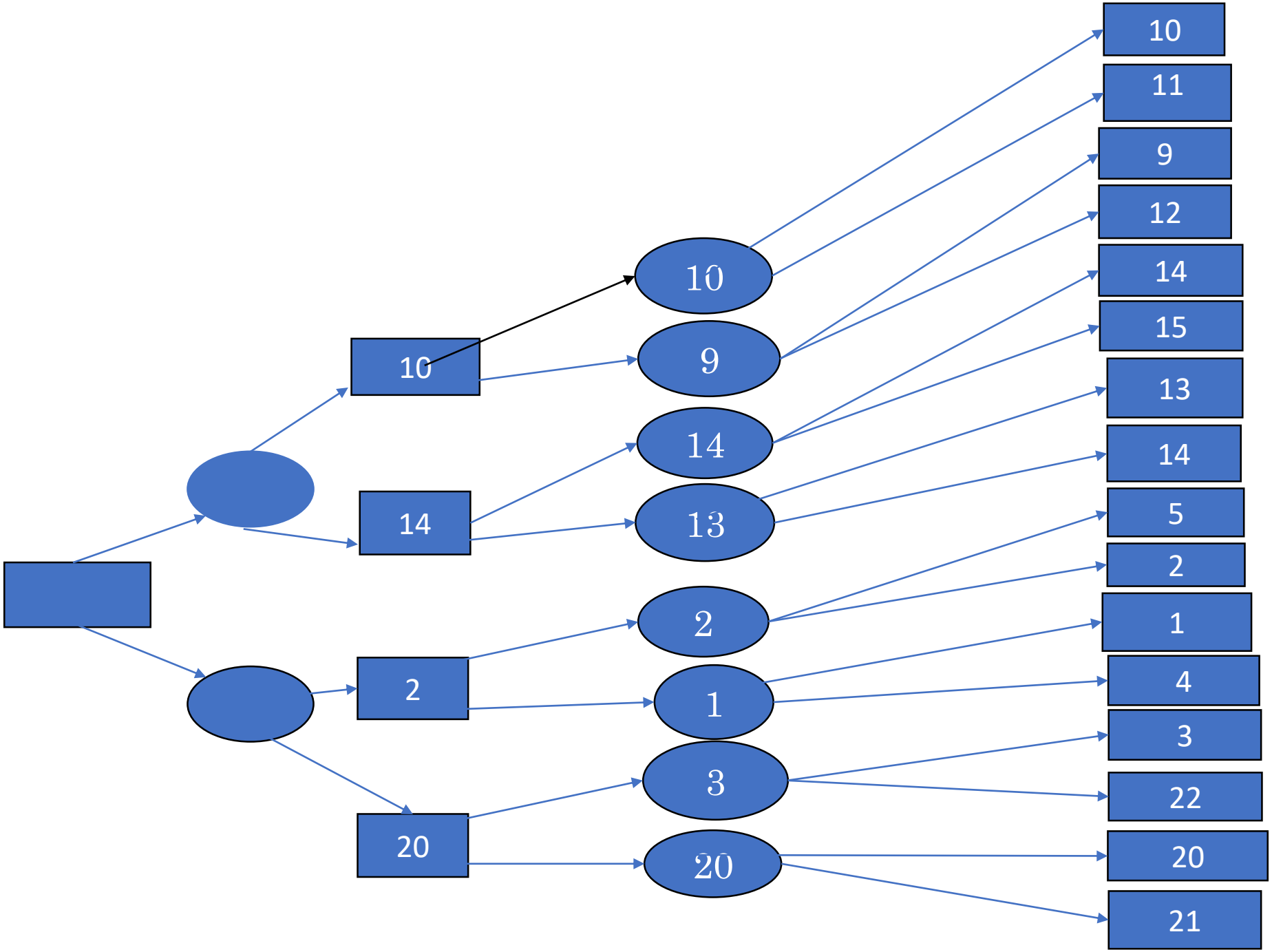
Confess Strongly Dominates over SILENT

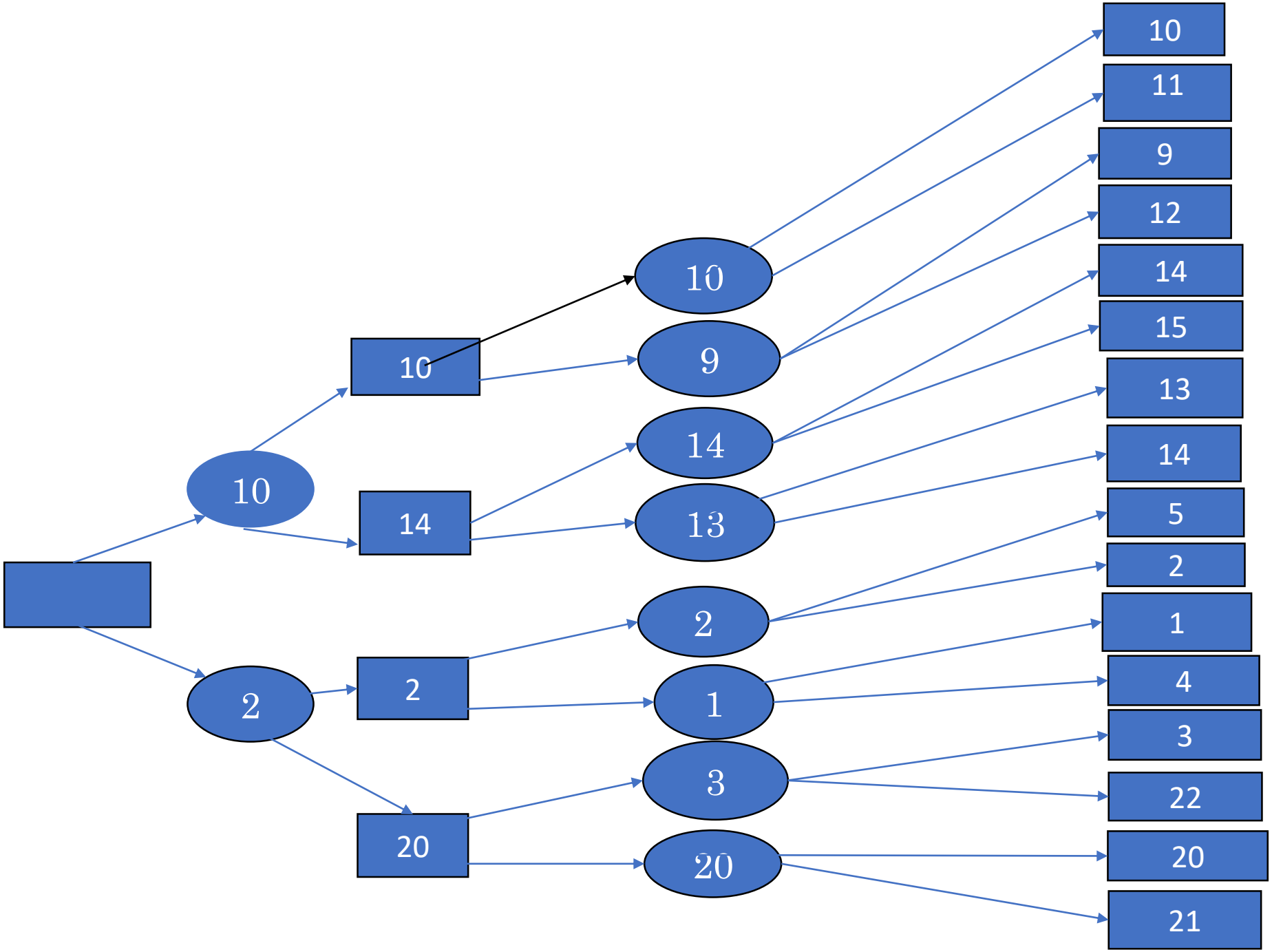
Another Example.....

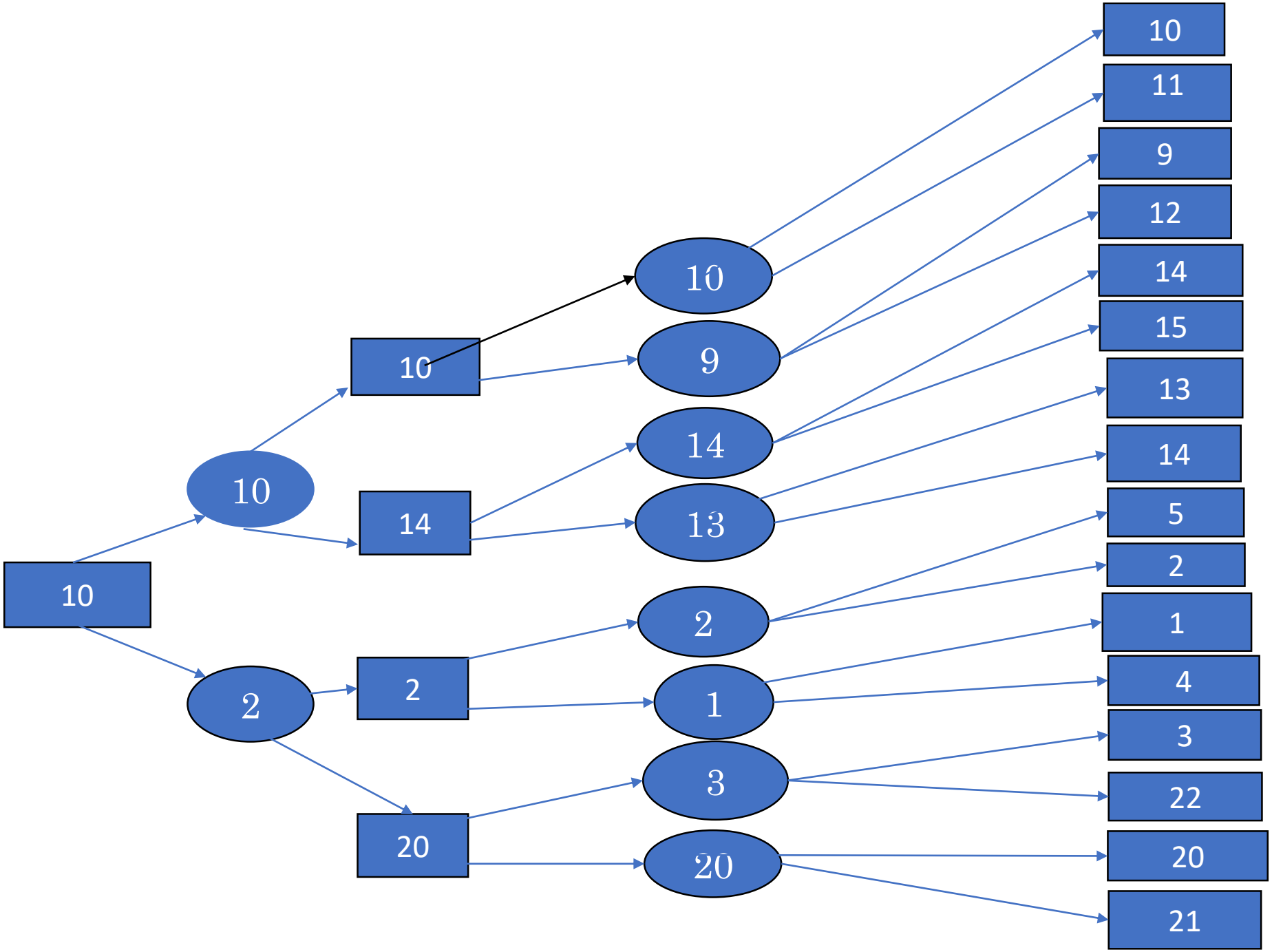
- Player A: Has to Choose an action
- Look off for A
- Choose min for B
- Choose max for A



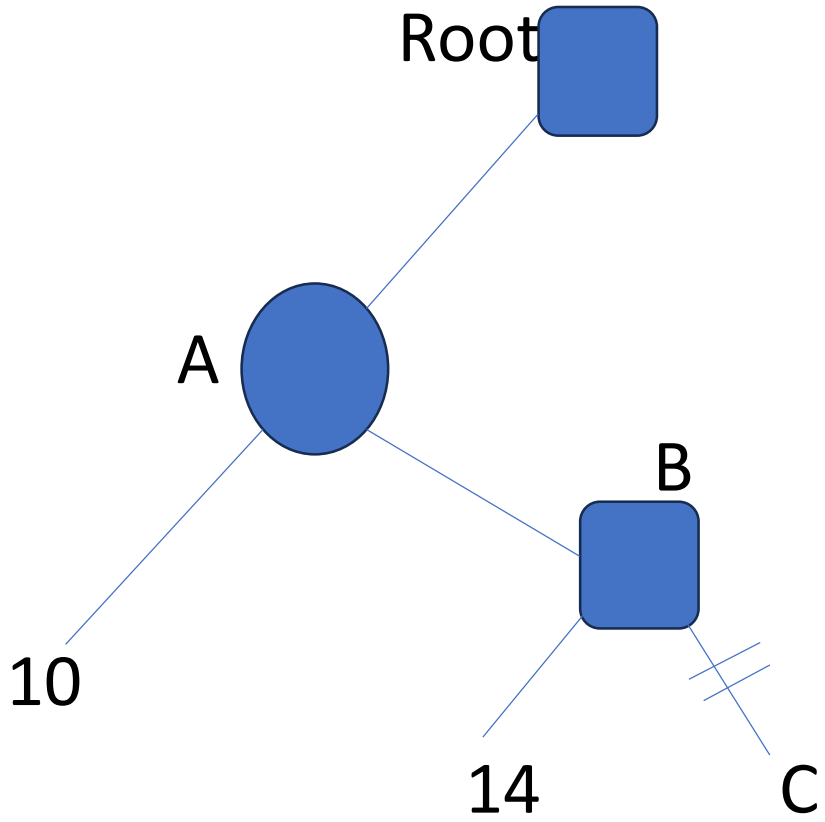

10
9
14
13
2
1
3
20
10
11
9
12
14
15
13
14
5
2
1
4
3
22
20
21


10
14
2
20
10
9
14
13
2
1
3
20
10
11
9
12
14
15
13
14
5
2
1
4
3
22
20
21


10
2
10
14
2
20
10
9
14
13
2
1
3
20
10
11
9
12
14
15
13
14
5
2
1
4
3
22
20
21


10
10
2
10
14
2
20
10
9
14
13
2
1
3
20
10
11
9
12
14
15
13
14
5
2
1
4
3
22
20
21

Shallow Cutoff



REFERECENCES

- Artificial Intelligence - A Modern Approach by Stuart Russell and Peter Norvig
- GeeksForGeeks - https://www.geeksforgeeks.org/
- Javatpoint - https://www.javatpoint.com/
- Scalar - https://www.scaler.com/

CONTRIBUTIONS

- 2201AI33 – Saksham Singh 20%
- 2201AI34 – Samar Kumar Srivastava 20%
- 2201AI35 – Saumya Pratap Singh 20%
- 2201AI36 – Somil Aggarwal 20%
- 2201AI37 – Sowmya Raj 20%