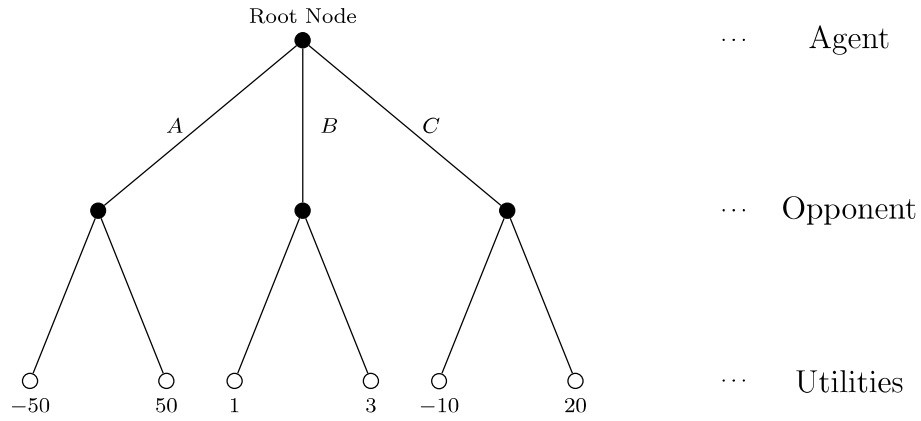
1 Intro



The game is as follows.

1. There are 3 containers and each container has 2 numbers in it.
2. There are 2 players, the agent and the opponent.
3. The agent choose the container first.
4. The opponent chooses a number from the container selected by the agent.
5. The final utility of the agent is the number picked by the opponent.

Utility: It is the amount of goodness/benefit that you get out of the game.

Suppose the opponent chooses to play in a stochastic manner such that he/she chooses the number uniformly at random from the given numbers. Then which container should the agent choose so as to get the **maximum utility** at the end of the game?
The expected utility by choosing the containers A, B and C are 0, 2, and 5 respectively. So, the agent should choose the container C.

2 Two Player Zero Sum Games

Two-player zero-sum games are a type of mathematical game in which one player's gain is exactly balanced by the other player's loss. The total utility available in the game is constant; hence, the sum of the gains and losses of all players is zero. In these games, the interests of the players are completely opposed, meaning that one player's win is the other's loss.
Examples: Chess, tic-tac-toe, checkers
Lets start with some of the basic terminologies for 2-player zero sum games:

- **players:** {Agent, Opponent}
- S_0: Starting state of the game

- **actions(s):** Possible actions at state s
- **player(s):** The player who has to make the move at state s
- **succ(s, a):** The resulting state if action a is taken at state s
- **isEnd(s):** Returns if the given state s i an end state
- **utility(s):** Agent's utility at an end state s

Example: Chess

- **players:** {White, Black}
- S_0: Starting state of the game
- **actions(s):** All legal actions move at state s by *player(s)*
- **player(s):** The player who has to make the move at state s
- **succ(s, a):** The resulting state if action a is taken at state s
- **isEnd(s):** Returns whether the given state s is a checkmate or draw
- $$\textbf{utility(s):} = \begin{cases} +M & \text{if white wins} \\ -M & \text{if black wins} \\ 0 & \text{if there is a draw} \end{cases}$$

  where M is a large positive number

3 Strategy of a player

The strategy of a player is the action we must take,if we end up in any of the state. It can be deterministic, probabilistic and also random. Now, we study about different types of strategies.
We represent the strategy of the player i at state s as π_i(s).

3.1 Deterministic

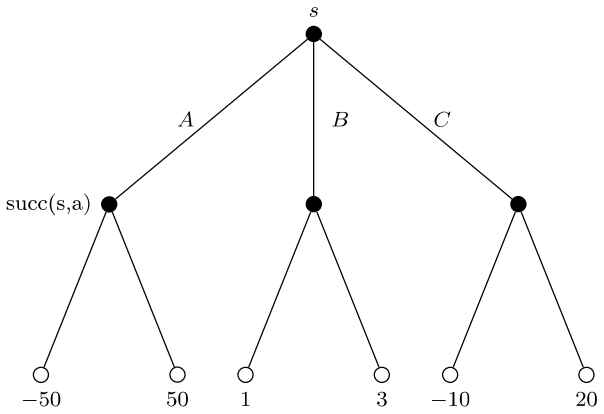
$$\pi_i(\text{s}) \ \epsilon \ \text{actions(s) if player(s)} = \text{i}$$

3.2 Randomized

$$\pi_i(\text{s}) \ \epsilon \ \Delta\text{actions(s)}$$

where Δactions(s) represent the probability distribution over the set of possible actions
Example: π_B(s) = (1/3,1/3,1/3) - The brother choose one out of the 3 possible actions uniformly at random

4 Games with Partial Information

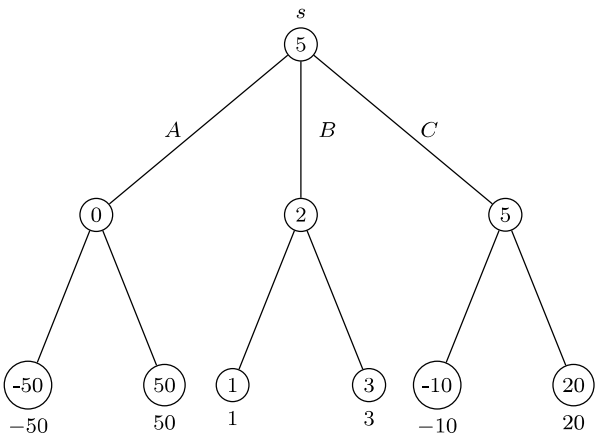


a = A

$$u_{\text{agent}}(S) = \begin{cases} \text{utility}(S) & \text{if isEnd}(S) \\ \sum_{a \in \text{actions}(S)} \pi_{\text{agent}}(S)[a] \cdot u_{\text{agent}}(\text{succ}(s,a)) & \text{if player}(S) = \text{agent}, \\ \sum_{a \in \text{actions}(S)} \pi_{\text{opponent}}(S)[a] \cdot u_{\text{agent}}(\text{succ}(s,a)) & \text{if player}(S) = \text{opponent}, \end{cases}$$

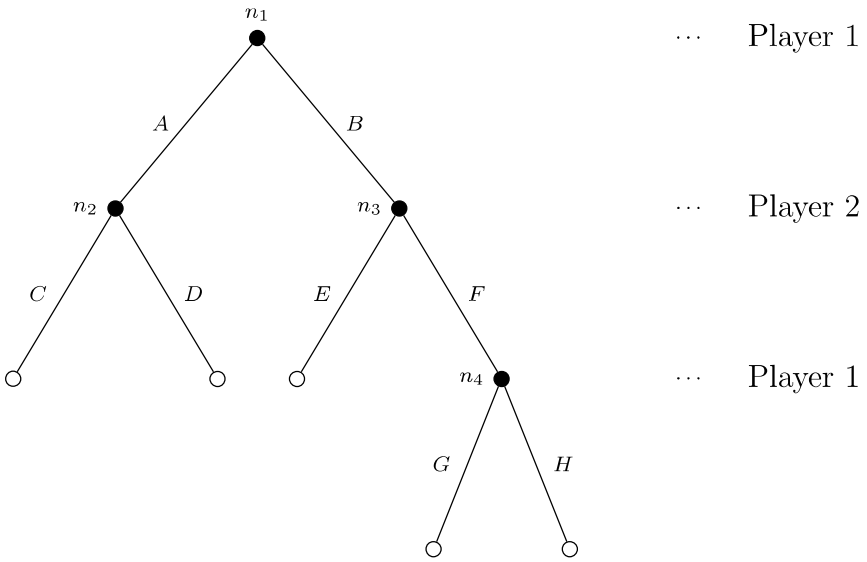
Example-1:

Opponent is stochastic and agent is utility maximizer(max player)



5 Subgame and subgame perfection

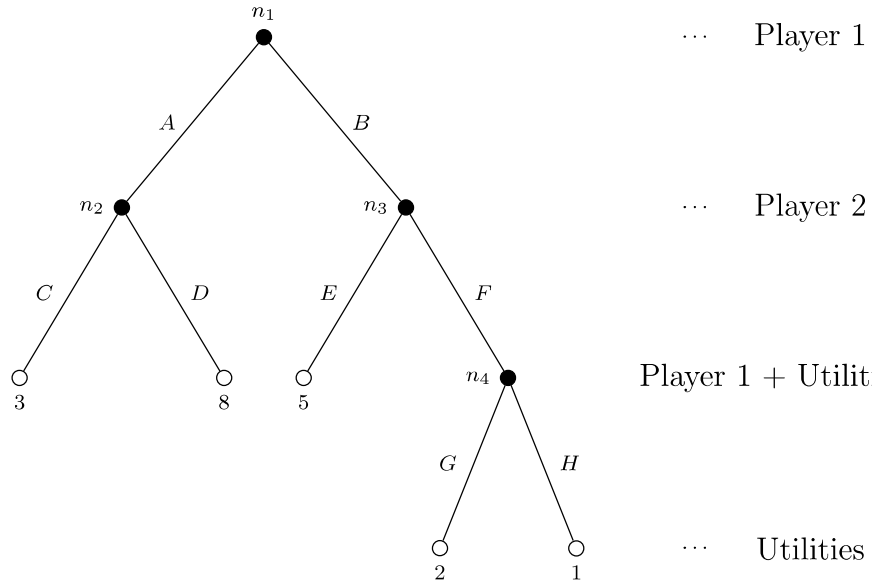
Let's take an example



Assume player 1 (agent) as a max player and player 2 (opponent) as a min player. So, their utilities will be

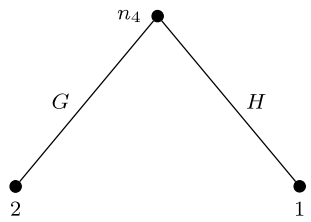
$$u_{\text{agent}}(S) = \begin{cases} \text{utility}(s) & \text{if isEnd}(S) \\ \max\left(u_{\text{agent}}(\text{succ}(s,a))\right) & \text{if player}(S) = \text{player 1}, \forall a \in \text{actions}(S) \\ \min\left(u_{\text{agent}}(\text{succ}(s,a))\right) & \text{if player}(S) = \text{player 2}, \forall a \in \text{actions}(S) \end{cases}$$

A subgame rooted at s is the restriction of the game at the subtree rooted at s, where isEnd(s) is false



We now traverse the gametree from below to top:

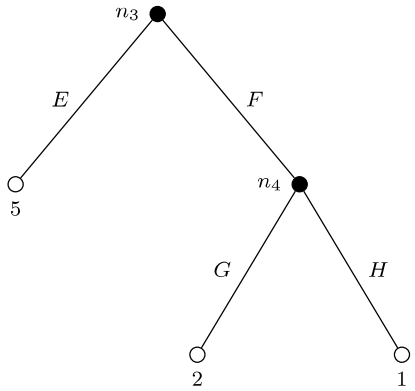
GameTree for n_4



$\cdots$ Player 1

As player 1 is a max player, he chooses the node with maximum utility, that is G

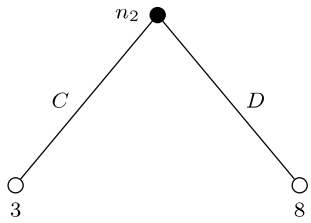
GameTree for n_3



$\cdots$ Player 2

We got the utility at n_4 to be 2. Player 2 is a min player, so he chooses node with minimum utility. Hence he chooses node F

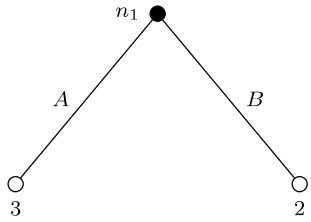
GameTree for n_2



$\cdots$ Player 2

Player 2 chooses node C for minimum utility

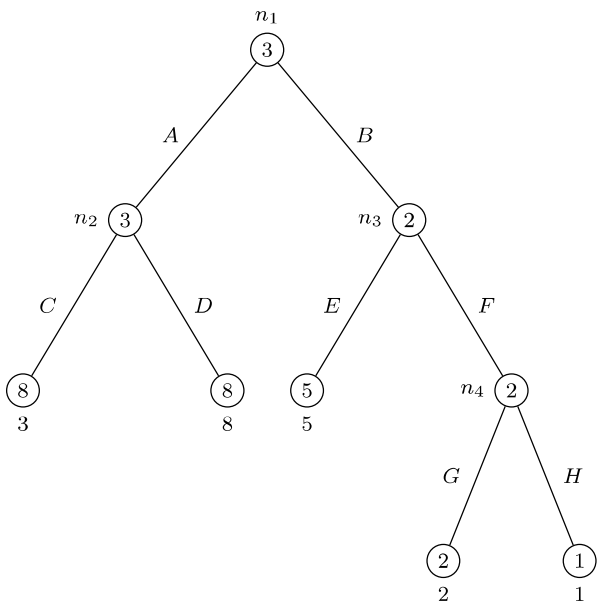
GameTree for n_1



$\cdots$ Player 1

We got the utility at n_3 to be 2 and n_2 to be 3. Now, player 1 chooses the node A for maximum utility

Now the final **GameTree** with utilities will be



Subgame perfect equilibrium is an equilibrium at every subgame. We can say that the above gametree with utilities is an equilibrium state because, given that player2 chooses a node, player1 cannot choose a better node than the above choices and vice-versa.

6 Backward Induction

It is an algorithm to analyse best strategy for a game, starting with last player's action and moving backward. Following is the pseudo code for this algorithm:

```
Function BackInd(s)
    if isEnd(s) then
        return U_agent(s), φ
    endif
    if player(s) = agent then
        bestUtil = −∞
        for all a ∈actions(s) do
            utilAtchild, bestAvect ←BackInd(succ(s,a))
            if utilAtchild > bestUtil then
                bestUtil = utilAtchild
                bestAvect = append(a,bestAvect)
            endif
    endif
    if player(s) = opponent then
        bestUtil = +∞
        for all a ∈actions(s) do
```

```
            utilAtchild, bestAvect ←BackInd(succ(s,a))
            if utilAtchild < bestUtil then
                bestUtil = utilAtchild
                bestAvect = append(a,bestAvect)
            endif
    endif
    return bestUtil, bestAvect
```

If we can use this algorithm and solve the Chess game (also Checkers and Go), then why are we still playing it? Because the number of nodes needed to construct GameTrees for Checkers, Chess and Go respectively are 10^{20}, 10^{40} and 10^{170} which are very large and the games cannot be solved any sooner using the Back Induction algorithm.