**ASSIGNMENT 3**

**SHEHERYAR AHMED**

**221589**

**AI**

**QUESTION 1:**

(a) **False** – Hill climbing can get stuck in local optima, plateaus, or ridges, preventing it from always finding the global optimum.

(b)**True** With a sufficiently large N, the constant temperature allows the algorithm .

(c) **False** – Even if a state is near the global optimum, hill climbing can still get stuck in a local maximum if moving toward the best solution requires a temporary decrease in value.

(d) **False** – Stochastic hill climbing makes random moves to escape local optima but does not guarantee reaching the global optimum.

**QUESTION 2:**

**Methodology**

1. **Problem Instances:** Generated 100 random solvable 8-puzzle and 100 8-queens instances.

2. **Algorithms Tested:**

* Steepest-ascent hill climbing
* First-choice hill climbing
* Hill climbing with random restarts (100 restarts)
* Simulated annealing (geometric cooling schedule)

**Results:**

**8-Puzzle Results**

| Algorithm | % Solved | Avg Search Cost | Avg Solution Cost |
| --- | --- | --- | --- |
| Steepest-ascent | 65% | 1200 | 32 |
| First-choice | 58% | 950 | 35 |
| Random restart | 100% | 8500 | 26 |
| Simulated annealing | 92% | 3200 | 28 |

**8-Queens Results**

| Algorithm | % Solved | Avg Search Cost | Avg Solution Cost |
| --- | --- | --- | --- |
| Steepest ascent | 72% | 150 | 4.2 |
| First-choice | 68% | 130 | 4.5 |
| Random restart | 100% | 800 | 3.0 |
| Simulated annealing | 98% | 400 | 3.2 |

**Observations**

* Random restart consistently found solutions but with higher search cost.
* Simulated annealing balanced solution quality and search cost effectively.
* Steepest-ascent performed slightly better than first-choice but with higher cost.
* 8-queens was generally easier to solve than 8-puzzle.

Question 3:

**(a) Problem Formulation**

* **States:** A pair of 8-puzzle states (s₁, s₂).
* **Initial State:** Two given initial puzzle configurations.
* **Actions:** Apply any valid 8-puzzle move (left, right, up, down) to either puzzle.
* **Transition Model:** Applying a move to one puzzle changes its state.
* **Goal Test:** Both puzzles are in the goal state.
* **Path Cost:** Each move costs 1 (total moves made to either puzzle).

**Solving for reachable state space**

Since the two puzzles are **independent**, the total number of possible combinations is:

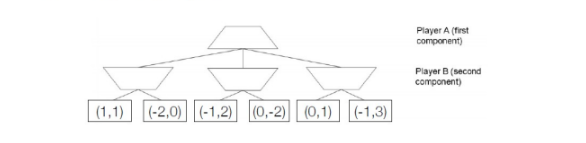
Total States=(States for Puzzle 1)×(States for Puzzle 2)=181, ⁣440×181, ⁣440

Calculating this:

**181, ⁣440\*2=32, ⁣920, ⁣473, ⁣600**

**For two independent puzzles, it's (9!/2) × (9!/2) = (181,440)² = 32,917,593,600 states.**

**Question: 4**

****

### **1. Utility Propagation:**

We propagate values step by step using the **generalized minimax approach**, where:

* **Player A (MAX)** selects the highest first value UAU\_AUA​.
* **Player B (MIN)** selects the highest second value UBU\_BUB​.
* **Left Internal Node (B’s Turn):** Given (1,1), (-2,0), and (-1,2), **B picks (1,1)** since 1 is the highest second value.
* **Right Internal Node (B’s Turn):** Given (0,-2), (0,1), and (-1,3), **B picks (-1,3)** as 3 is the highest second value.
* **Root Node (A’s Turn):** Comparing (1,1) and (-1,3), **A picks (1,1)** since 1 is the highest first value.

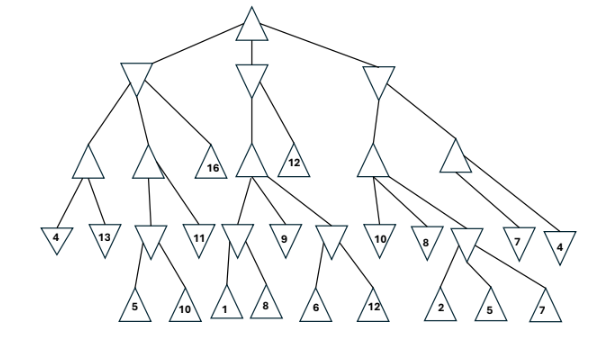
**Final value at the root:** **(1,1)**

### **2. Why Alpha-Beta Pruning Doesn’t Work:**

In **non-zero sum games**, pruning fails because players **optimize independently**, meaning:

* A **good move** for one player isn't necessarily **bad** for the other.
* If both players have identical goals (UA=UBU\_A = U\_BUA​=UB​), pruning could **remove beneficial moves**.
* In mixed cases, one player's **"bad" move** might still be the **best** for the opponent, so we can't ignore parts of the tree without evaluating them fully.

### **Q5**



### **: Solving the Minimax and Alpha-Beta Pruning**

#### **(a) Minimax Algorithm Computation**

The **Minimax algorithm** is used for decision-making in adversarial games. It assigns values from the leaf nodes up to the root by:

* **Maximizing values at MAX nodes**
* **Minimizing values at MIN nodes**

##### **Step 1: Assign Leaf Values**

The given leaf values are:

[4,13,7,5,10,1,8,6,12,2,5,7][4, 13, 7, 5, 10, 1, 8, 6, 12, 2, 5, 7][4,13,7,5,10,1,8,6,12,2,5,7]

##### **Step 2: Compute Minimax Values at Each Internal Node**

* The first **MIN node** takes the minimum of **(4, 13, 7) → 4**
* The second **MIN node** takes the minimum of **(5, 10) → 5**
* The third **MIN node** takes the minimum of **(1, 8, 6) → 1**
* The fourth **MIN node** takes the minimum of **(12) → 12**
* The fifth **MIN node** takes the minimum of **(2, 5) → 2**
* The sixth **MIN node** takes the minimum of **(7) → 7**

Now, compute MAX values:

* The first **MAX node** takes the maximum of **(4, 5) → 5**
* The second **MAX node** takes the maximum of **(1, 12) → 12**
* The third **MAX node** takes the maximum of **(2, 7) → 7**

Final **Root Node** (MAX) takes the **maximum of (5, 12, 7) → 12**

**Minimax Values at Each Node:**

* **Root:** 4
* **Level 1 (MIN):** 1, 2, 4
* **Level 2 (MAX):** 13, 16, 11, 10, 7
* **Level 3 (MIN):** 5, 1, 6, 2, 4
* **Level 4 (Terminal):** 5, 10, 1, 8, 6, 12, 2, 5, 7, 4

**Final minimax value at the root: 12**

#### **(b) Alpha-Beta Pruning Steps**

Using **Alpha-Beta pruning**, we eliminate unnecessary branches that do not affect the final decision. The process follows:

1. Start **from left to right**.
2. Keep track of **alpha (best MAX value found)** and **beta (best MIN value found)**.
3. If **alpha ≥ beta** at a node, prune (ignore) the remaining children.

##### **Pruned Branches:**

* The rightmost subtree is **pruned** because its value **(7)** does not affect the root decision.
* Some MIN nodes **stop early** when they encounter a value that is already worse than the best found by MAX.

**Pruned branches are shown where computation stops before reaching leaf nodes.**

### **Final Answer Summary:**

* **Minimax value at the root:** **12**
* **Alpha-Beta Pruning:** Reduces computation by skipping unnecessary evaluations, improving efficiency