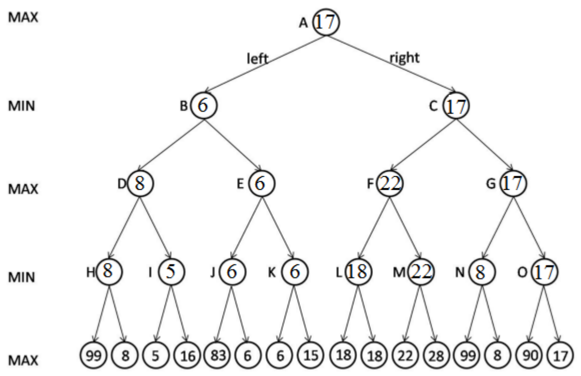
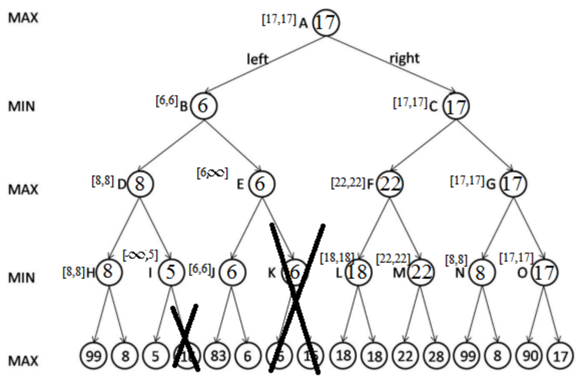
Problem 1

Problem 2

- A.



- B.



We cut off the subtree at node K since node B is a MIN and will only accept a value less than 8 we know that we wont get a value greater than 6 down said subtree. The same also occurs below node I. The terminal node 15 is pruned because node D will accept any number greater than 8 however, node I is a MIN and the left terminal node being 5 allows us to denote that the right terminal node cannot be less than 5.

- C. The max player will choose 17 at the root state by using the exhaustive Minimax Algorithm. The same result will occur when using alpha-beta pruning and the max player will again choose 17. The most optimal move is guaranteed using either the exhaustive Minimax algorithm or the alpha-beta pruning algorithm. This is accomplished by alpha-beta pruning ignoring subtrees that are irrelevant while exhaustive Minimax algorithm checks every node regardless of relevance. The outcome is the same however, since alpha-beta pruning ignores irrelevant subtrees it is more efficient.

- D.

- E.

Problem 3

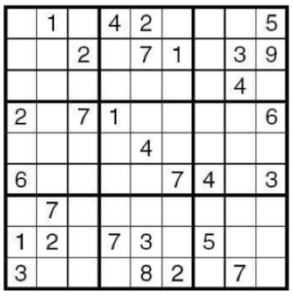
A. **Variables:** Defined as $n_{i,j}$ where $i \in$ {set of all row numbers} and $j \in$ {set of all column numbers} for each cell n on the board. Essentially, each square on the sudoku board will be a variable, for a total of 81 variables.

Domain: Defined as the set of possible values in sudoku. Here, this set is defined as $\{1, 2, ..., 9\}$

Constraints:

1. All variables in any row must be assigned distinct values.
2. All variables in any column must be assigned distinct values.
3. All variables in any 3x3 region must be assigned distinct values.
4. The assignments given to variables in the start state cannot be changed.

B. **Start State:** The start state is a given board that has variables assigned with certain values. These values are the basis for the constraints for the rest of the problem and may not be changed. Below is an example start state.



	1		4	2				5
		2		7	1		3	9
							4	
2		7	1					6
				4				
6					7	4		3
	7							
1	2		7	3		5		
3				8	2		7	

Successor Function:

1. Select a random unassigned variable
2. Assign a value to this variable that does not violate any constraints
3. The resulting state is the successor

C.

D. Pseudocode for Local Search

Algorithm 1 Sudoku Local Search Algorithm

```
procedure SudokuSolver(Board board)
    board ← randomizeCompleteState(board)
    while !completeAndSatisfied(board) do
        nextBoard ← updateState(board)
        if evaluateSatisifiability(nextBoard) > evaluateSatisifiability(board) then
            board ← nextBoard
    return board
```

Problem 4

For Superman to be defeated, it has to be that he is facing an opponent alone and his opponent is carrying Kryptonite. Acquiring Kryptonite, however, means that Batman has to coordinate with Lex Luthor and acquire it from him. If, however, Batman coordinates with Lex Luthor, this upsets Wonder Woman, who will intervene and fight on the side of Superman.

- A.

  Superman defeated = S

  Facing Opponent Alone = A

  Opponent carrying Kryptonite = K

  Batman coordinates with Lex Luther = B

  Wonder Woman Upset = W

- B. Superman being defeated implies that not only did his opponent have Kryptonite but that Superman was also fighting alone. This is equivalent to Superman not being defeated or his opponent has Kryponite and Superman is fighting alone. This all equates to:

  $$(\neg S \vee K) \wedge (\neg S \vee A)$$

  Superman fighting alone implies that Superman was defeated.

  $$(\neg K \vee B)$$

  Batman coordinating with Lex Luther implies that Wonder Woman will be upset and will team up with Superman.

  $$(\neg B \vee W)$$

  Wonder Woman being upset implies that she will team up with Superman and he will not be fighting alone.

  $$(\neg W \vee \neg A)$$

- C. Based off the knowledge presented above we can prove that Batman cannot defeat Superman.

  $$\begin{gathered}(\neg S \vee K) \wedge (\neg S \vee A) \wedge (\neg K \vee B) \wedge (\neg B \vee W) \wedge (\neg W \vee \neg A)\\ (\neg S \vee \neg S) \wedge (\neg A \vee A) \wedge (\neg K \vee K) \wedge (\neg B \vee B) \wedge (\neg W \vee W)\\ \neg S\end{gathered}$$

Problem 5

- A.

Problem 6

- A. Using the minimum value of $h_1(n)$ and $h_2(n)$ for each state:

  $$\begin{gathered}\text{Given that } h_1(n) \leq h^*(n) \text{ is always true,}\\ Min(h_1(n), \infty) \leq h^*(n)\\ \text{Therefore } h_1(n) = minimum(h_1(n), h_2(n)) \text{ is admissible.}\\ \text{consistent?}\end{gathered}$$

- B. Using the maximum value of $h_1(n)$ and $h_2(n)$ for each state:

$$\text{Given that } h_1(n) \leq h^*(n)$$
$$h_2(n) \leq h^*(n)$$
$$Max(h_1(n),\, h_2(n)) \leq h^*(n)$$
$$\text{Therefore } h_4(n) = maximum(h_1(n),\, h_2(n)) \text{ is admissible.}$$
$$\text{consistent?}$$

- C. The defined heuristic function

  $$h_3(n) = w \times h_1(n) + (1 - w)h_2(n), \text{ where } 0 \leq w \leq 1$$

  is admissible only when w is a value less than 0.5, any value larger than 0.5 will cause $h_1(n)$ to be multiplied by a value larger than $h_2(n)$. Since the the bounds are from 0 to 1 the heuristic is NOT admissible.

  consistent?

- D. Considering the informed, best-first search algorithm with an object function of $f(n) = (2 - w) \times g(n) + w \times h(n)$. It is guaranteed to work for $0 \leq w \leq 1$ since being multiplied by g(n) which is a constant implies no effect on which order chosen paths are arranged. However, if $w > 1$, then the goal state may have an overestimated distance which will make the heuristic not admissible and so not optimal.

  Given the information above we can assess the function for when the values of w are 0, 1, and 2.

  $w = 0$:

  When $w = 0$ it will make $f(n) = 2g(n)$ and causes the algorithm to perform like a Uniform Cost Search since there is no weight assigned it will find the most optimal path however, it will not be efficient.

  $w = 1$:

  When $w = 1$ it will make $f(n) = g(n) + h(n)$ which causes the algorithm to perform like A^* and as with $w = 0$ is guaranteed to find the optimal path however, is more efficient.

  $w = 2$:

  When $w = 2$ it will make $f(n) = 2h(n)$ which causes the algorithm to perform like Greedy Best first search.