1. (18 points) Genetic Algorithms

This question investigates using genetic algorithms to solve a problem. The problem considered is to generate a target phrase from a population of random strings (problem and code selected from aima-python code repository[1] and examples).

For this example, the phrase to be generated is "4811 + AI +GAs = fun!". We also will need to define the gene pool to consist of any string consisting of the upper and lower case English alphabet, digits 0-9, and a few puctuation characters - { space ! . + - , ?}. Finally, several parameters must be given for the genetic algorithm:

- `ngen` - maximum number of generations the algorithm will run
- `max_population` - the size of the population in each generation
- `mutation_rate` - the rate of mutation (value between 0 and 1).
- `f_thres` - the fitness threshold used to determine if the target phrase is found to terminate the algorithm

For this problem, the fitness function evaluates strings by the number of characters matching the target phrase:

```
def fitness_fn(sample):
fitness = 0
    for i in range(len(sample)):
        if sample[i] == target[i]:
            fitness += 1
    return fitness
```

Begin, by opening the Python notebook linked from the canvas assignment `w2_q2.py`. The notebook begins with a short introduction to Python notebook then has code to run `genetic_algorithm_stepwise` the GA method that reports for each generation, the best individual (highest fitness) and the fitness function value. The method is run with the default configuration: `ngen=1000`, `max_population=100`, `mutation_rate=0.07`, and the `f_thres=len(target)`.

You are asked to explore the parameter space by varying:

(a) the number of generations, `ngen`, between 50 and 500 in increments of 50 while keeping all other parameters fixed at the default values.

(b) the population size, `max_population`, between 50 and 500 in increments of 50 while keeping all other parameters set to the defaults.

(c) the mutation rate, `mutation_rate`, between 0.025 and 0.25 in increments of 0.025 while keeping all other parameters set to the defaults.

Report your findings in two figures plotting execution time and fitness score of best individual vs. the parameter being varied. Results should be reported as an average and include errorbars for standard deviation over at least 5 runs of the GA. Comment on the results.

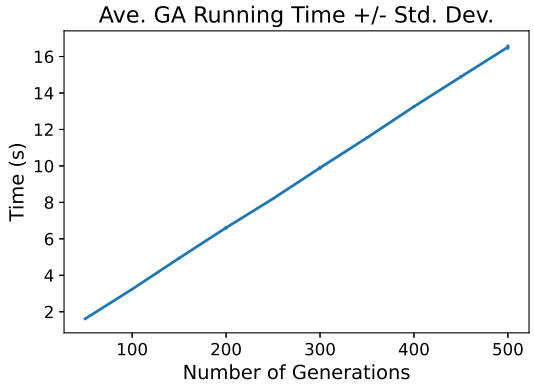
Code to implement part (a), including creating the plots, is provided as an example. You can use the plots but still need to add your own comments on the results.

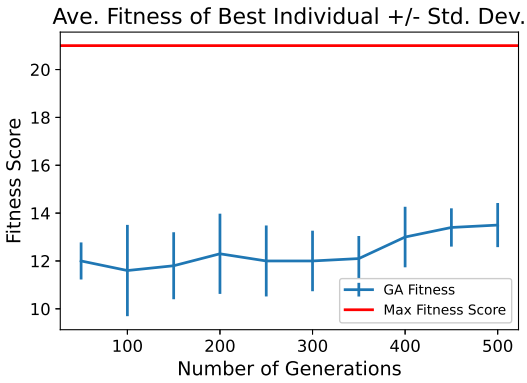
The plots generated in the Python notebook can be downloaded for inclusion in your solution to this assignment.

[1] `https://github.com/aimacode/aima-python`

Solutions:

(a) the number of generations, `ngen`, between 50 and 500 in increments of 50





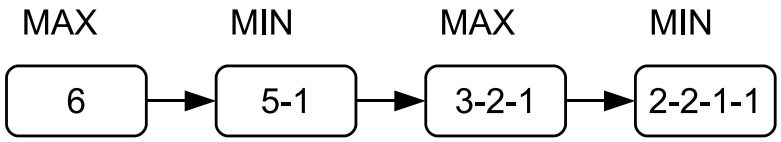
(b) the population size, `max_population`, between 50 and 500 in increments of 50

(c) the mutation rate, `mutation_rate`, between 0.025 and 0.25 in increments of 0.025

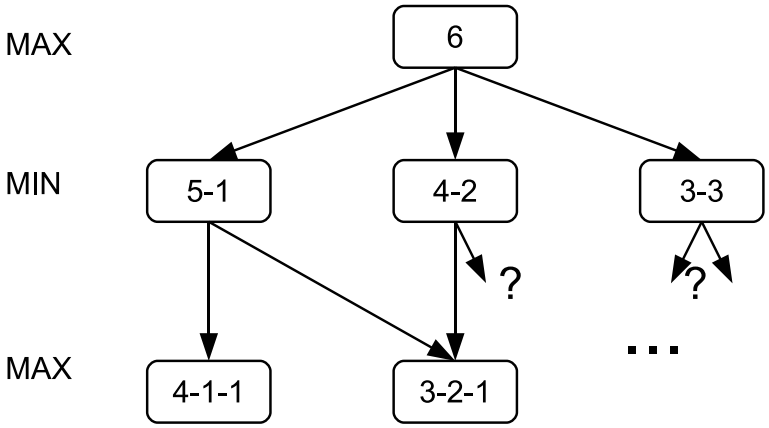
2. (8 points) Games

Consider the game *nim* with the following rules. The game starts with n sticks in a pile. Turns alternate between players. For each turn, the active player must select a pile and divide it into two smaller piles. Piles containing only one or two sticks can not be divided further. The last player able to make a legal move wins. Note, this is one variation of the game, many others exist.

Consider the following path with $n = 6$. MAX is the starting player, then MAX and MIN alternate actions with MAX winning (last player able to make a legal move).



The following is part of the game tree for $n = 6$.



You will examine the game tree for *nim* with $n = 7$.

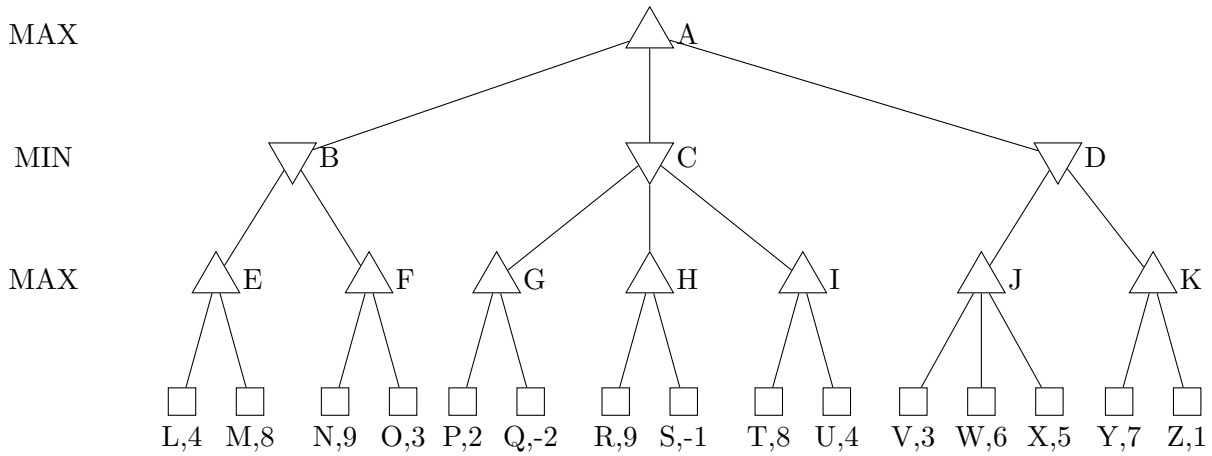
Solutions:

(a) (6 points) Draw the entire game tree for *nim* with $n = 7$ and assuming MAX is the starting player, following the partial game tree above for construction guidelines. Each level of the tree should correspond to either MIN or MAX's moves. Label terminal nodes with the utility of 1 if it is a win for MAX, or -1 if it is a win for MIN. Run the Minimax algorithm and report the minimax values besides each node of the tree.

(b) (1 point) Who will win this game of *nim*?

(c) (1 point) Can any guarantees on the outcome of the game be stated (i.e., MAX always wins, MIN always wins, the first player never wins, etc.)?

3. (24 points) Adversarial Search



Consider the game tree above:

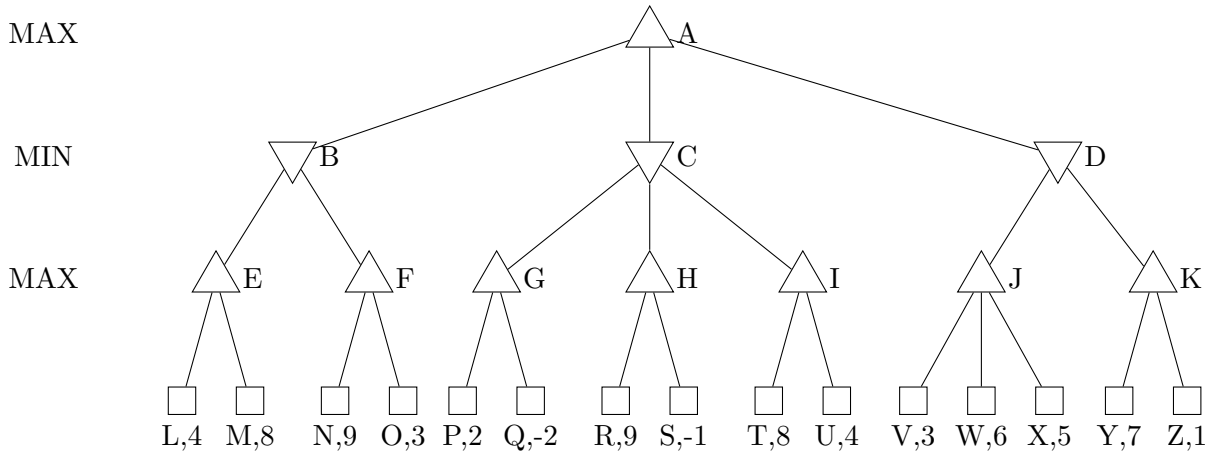
(a) (4 points) Use the Minimax algorithm to compute the minimax value at each node for the game tree.

(b) (10 points) Show the application of alpha-beta pruning on the same game tree (copy available below) expanding successors from left to right. Indicate where the pruning occurs with an "X". Also, record the $[\alpha, \beta]$ pair for each node when first visited (to the left of the node), then final values (to the right of the node).

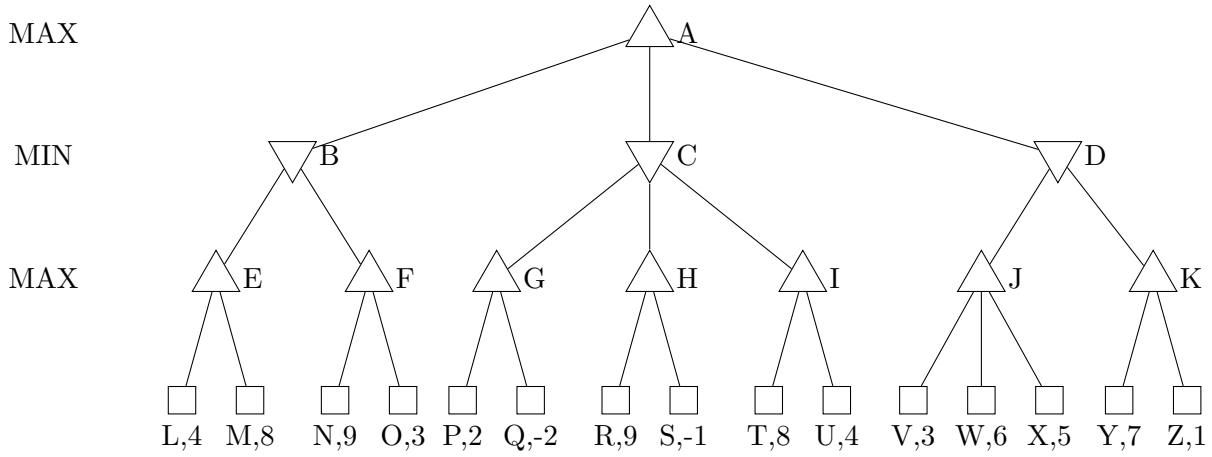
(c) (10 points) Repeat (b) expanding successors from right to left.

Solutions:

(a)



(b) Left to Right
The $[\alpha, \beta]$ pair values given on the left of each node are those when the node is first visited, the pair on the right is the final values.



(c) Right to Left
The $[\alpha, \beta]$ pair values given on the left of each node are those when the node is first visited, the pair on the right is the final values.