COMP 472	**Assignment 3**	**Winter 2021**
Due dates:	**April 19, 2021**	
Late submission:	No late submission	
Teams:	You can do the assignment individually or in teams of 4 students.	
Purpose:	The purpose of this assignment is to implement and analyze **adversarial search** for a game.	

1. The *PNT-Game*

In this assignment you will implement the Alpha-Beta pruning algorithm to play a two-player game called *PNT*: pick numbered-tokens.

1.1 Rules of the game

The game starts with n numbered-tokens 1, 2, 3, ..., n. Players take turns removing one of the remaining numbered-tokens. At a given turn there are some restrictions on which numbers (i.e., tokens) are legal candidates to be taken. The restrictions are:

- At the first move, the first player *must* choose an odd-numbered token that is strictly less than $n/2$.

  For example, if $n = 7$ $(n/2 = 3.5)$, the legal numbers for the first move are 1 and 3.

  If $n = 6$ $(n/2 = 3)$, the only legal number for the first move is 1.
- At subsequent moves, players alternate turns. The token number that a player can take must be a ***multiple or factor*** of the last move (note: 1 is a factor of all other numbers). Also, this number may ***not*** be one of those that has already been taken. After a token is taken, the number is saved as the new last move. If a player ***cannot*** take a token, he/she loses the game.

1.2 Example

Here is an example, to better understand the rules of the game, for the *PNT* game with $n = 7$:

```
Player A: 3
Player B: 6
Player A: 2
Player B: 4
Player A: 1
Player B: 7

Winner: Player B
```

2. Your Tasks

Implement a solution for the *PNT* game using Alpha-Beta pruning algorithm that obey the pseudocode described in Figure 5.7 of the textbook, page 154. The game has 2 players: player A (called Max) and player B (called Min). For a new game (i.e., no tokens have been taken yet), the Max player always plays first. Given a specific game board state, your program should ***compute the best move for the current player*** using the given heuristic static board evaluation function. That is, only a *single* move is computed.

2.1 Programming Environment

To program the assignment, you must use Python 3.8. In addition, you must use GitHub (make sure your project is private while developing).

2.2 The Input

A sequence of positive integers given as command line arguments separated by spaces:

PNT Player <#tokens> <#taken_tokens> <list_of_taken_tokens> <depth>

- #tokens: the total number of tokens in the game
- #taken_tokens: the number of tokens that have already been taken in previous moves. If this number is 0, this is the first move in a game, which will be played by Max. (Note: If this number is even, then the current move is Max's; if odd, the current move is Min's)
- List_of_taken_tokens: a sequence of integers indicating the indexes of the already-taken tokens, ordered from first to last token taken. Hence, the last token in the list was the token taken in the last move. If #taken_token is 0, this list will be empty. You can safely assume that these tokens were taken according to the game rules.
- depth: the search depth. If depth is 0, search to end game states (i.e., states where a winner is determined).

For example, the input with:

PNT Player 7 2 3 6 0

we have 7 tokens while 2 tokens, numbered 3 and 6, have already been taken, so it is the Max player's turn (because 2 tokens have been taken), and you should search to end game states.

There is no need to check the validity of the input sequence, you can assume that it will be correct.

2.3 The Output

You are required to print out the following information to the console, after your alpha-beta search has completed:

- The best move (i.e., the tokens number that is to be taken) for the current player (as computed by your alpha-beta algorithm)
- The value associated with the move (as computed by your alpha-beta algorithm)
- The total number of nodes visited
- The number of nodes evaluated (either an end game state or the specified depth is reached)
- The maximum depth reached (the root is at depth 0)
- The average effective branching factor (i.e., the average number of successors that are *not* pruned)

For example, here is sample input and output when it is Min's turn to move (because 3 tokens have previously been taken), there are 4 tokens remaining (3 5 6 7), and the Alpha-Beta algorithm should generate a search tree to maximum depth 3. Since the last move was 2 before starting the search for the best move here, only one child is generated corresponding to removing token 6 (since it is the only multiplier of 2). That child node will itself have only one child corresponding to removing token 3 (since it is the only factor of 6 among the remaining tokens). So, the search tree generated will have the root node followed by taking 6, followed by taking 3, which leads to a terminal state (Max wins). So, returning from these nodes, from leaf to root, we get the output below.

<u>Input:</u>

```
TakeTokens 7 3 1 4 2 3
```

Output:

```
Move: 6
Value: 1.0
Number of Nodes Visited: 3
Number of Nodes Evaluated: 1
Max Depth Reached: 2
Avg Effective Branching Factor: 1.0
```

Note: All doubles/floats should be printed with 1 decimal place.

2.5 Static Board Evaluation

The static board evaluation function should return values as follows:

- At an end game state where Player A (MAX) wins: 1.0
- At an end game state where Player B (MIN) wins: -1.0
- Otherwise,
  - if it is Player A (MAX)'s turn:
    - If token 1 is not taken yet, return a value of 0 (because the current state is a relatively neutral one for both players)
    - If the last move was 1, count the number of the possible successors (i.e., legal moves). If the count is odd, return 0.5; otherwise, return-0.5.
    - If last move is a prime, count the multiples of that prime in all possible successors. If the count is odd, return 0.7; otherwise, return-0.7.
    - If the last move is a composite number (i.e., not prime), find the largest prime that can divide last move, count the multiples of that prime, including the prime number itself if it hasn't already been taken, in all the possible successors. If the count is odd, return 0.6; otherwise, return-0.6.
  - If it is Player B(MIN)'s turn, perform the same checks as above, but return the opposite (negation) of the values specified above.

2.6 Other useful information

- Set the initial value of alpha to be `Double.NEGATIVE_INFINITY` and beta to be `Double.POSITIVE_INFINITY`
- To break ties (if any) between child nodes that have equal values, pick the smaller numbered token. For example, if tokens 3 and 7 have the same value, pick token 3.
- Your program should run within 10 seconds for each test case.

Note: There are three test cases posted with this assignment to help verify your implementation. The `testcase.txt` file contains the three cases, and the correct outputs are in files `output[1-3].txt`

Once your code is running, you will need to analyze and compare the performance of the game using 10 random testcases.

3. Deliverables

The submission of the assignment will consist of 2 deliverables: the code and a demo.

3.1 The Demos

You will have to demo your code to the TAs. Regardless of the demo time, you will demo the program that was uploaded as the official submission on or before the due date. The schedule of the demos will be posted on Moodle. The demos will consist in 2 parts: a small presentation of your program and a Q/A.

3.1.1 Presentation of your analysis

Prepare a few slides for a 5 minutes presentation explaining your Alpha-Beta program, the analysis (see section 2.4 and 2.5)

3.1.2 Q/A

No special preparation is necessary for the Q/A part of the demo. Your TA will give you more test cases and you will run your algorithm on this testcases. You will generate the corresponding outputs
In addition, your TA will ask each student questions on the code/assignment, and the student will be required to answer the TA satisfactorily. Hence every member of team is expected to attend the demo. Your individual grade will be a function of your individual Q/A (see Section 4).

4. Evaluation Scheme

Students in teams can be assigned different grades based on their individual contribution to project. Individual grades will be based on:

a) a peer-evaluation done after the submission.
b) the contribution of each student as indicated on GitHub.
c) the Q/A of each student during the demo.

The team grade will be based on:

Code	functionality, correctness and format of input and output, design and programming style,…	**70%**
Test cases	format, correctness,…	**5%**
Demo- QA	clear answers to questions, knowledge of the program, . . .	**10%**
Demo- Presentation	clarity and conciseness, the analysis, presentation skills, Time management . . .	**15%**
Total		**100%**

5 Submission

If you work in a team, identify one member as the team leader. The only additional responsibility of the team leader is to upload all required files from her/his account and book the demo on the Moodle scheduler. If you work individually, by definition, you are the team leader of your one-person team.

Submission Checklist

on GitHub:

- In your GitHub project, include a README.md file that contains on its first line the URL of your GitHub repository, as well as specific and complete instructions on how to run your program.

on Moodle:

- Create a PDF of your slides, and name your slides 472 Slides3 ID1 ID2 ID3 ID4.pdf where ID1 is the ID of the team leader.
- Create one zip file containing all your code, your file of 10 test cases, the corresponding output files and the README.md file. Name this zip file 472 Code3 ID1 ID2 ID3 ID4.zip where ID1 is the ID of the team leader.
- Zip 472 Slides3 ID1 ID2 ID3 ID4.pdf and 472 Code3 ID1 ID2 ID3 ID4.pdf and name the resulting zip file: 472 Assignment3 ID1 ID2 ID3 ID$.zip where ID1 is the ID of the team leader.
- Have the team leader upload the zip file on Moodle Assignment 3 submission.

Have fun!