UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG
School of Computer Science & Applied Mathematics

Advanced Analysis of Algorithms
2021 Assignment: Part 4

1 Introduction

The previous parts have allowed us to setup a position, generate the valid moves, and execute one of them. We can also determine whether a move results in the game being over. This is all we need to implement our AI search algorithms! In this section, we will implement a minimax agent, that can take in a position and output the move it wants to play.

2 Evaluation Function

Because we will not be able to search to the end of the game (the search tree is too large), we must use an evaluation function. This will allows us to search N moves into the future and evaluate the resulting position. An evaluation function is simply a function that accepts as input a board position, and returns a value that measures how good the position is for whoever it is to play. Importantly, the evaluation function is static — it does not consider moves into the future. The first submission will have us create a simple evaluation function, but they can as complex as required.

3 Search Algorithm

Once we have our evaluation, implementing the search function is fairly standard. In this submission, we will implement minimax, but an extension to $\alpha\beta$ pruning is not too difficult.

4 Input Hint

Hint: a reminder not to be careful when mixing `cin` with `getline`. An example of doing so is below:

```
int N;
cin >> N;
cin.ignore(); //NB!
for (int i = 0; i < N; ++i) {
    string fen;
    getline(cin, fen);
}
```

Submission 1: Evaluation

Write a C++ program that accepts a FEN string, sets up the board, and then outputs an evaluation of the position from the perspective of the player to move next. Our evaluation function here will be relatively simple, and is computed asd described below.

First, we compute the *raw score* according to the following rules:

1. If the board contains only a black and white lion and no other pieces, then it is a draw and the raw score is 0.
2. Otherwise, if the black lion is missing, then white has won and the raw score is 10000.
3. Otherwise, if the white lion is missing, then black has won and the raw score is -10000.
4. Otherwise, compute the value of the white pieces on the board according to the Piece Value table below. Then, do the same for the black pieces. In both cases, exclude the lions. The raw score is then the total value of white pieces minus the total value of black pieces.

The raw score above is the score from white's point of view. But perhaps it is black to play! Therefore, to compute the final score, we must consider whose turn it is. If it is white to play, multiply the raw score by 1. Otherwise, multiply it by -1. This is the final value that should be returned by our evaluation function.

Piece	Value
Pawn	100
Elephant	200
Zebra	300

Table 1: The relative value of a piece for both sides. For example, if white has two pawns and one elephant, they would have a total of $2 \times 100 + 200 = 400$, and if black has five pawns and one zebra, they would have a value of 1300.

Input

The first line of input is N, the number of input positions given as FEN strings. N lines follow, with each line containing a single FEN string.

Output

For each FEN string, output the evaluation of that position according to the above evaluation function.

Example Input-Output

Sample Input

```
3
2ele1z/ppppppp/7/7/7/PPP1PPP/2ELE1Z w 4
1z5/pPp1lP1/5ep/4P1e/4L1p/2p2pP/7 b 12
1z5/pPp1lP1/5ep/4P1e/4L1p/2p2pP/7 w 12
```

Sample Output

```
-100
900
-900
```

Visualisation of Above Test Cases

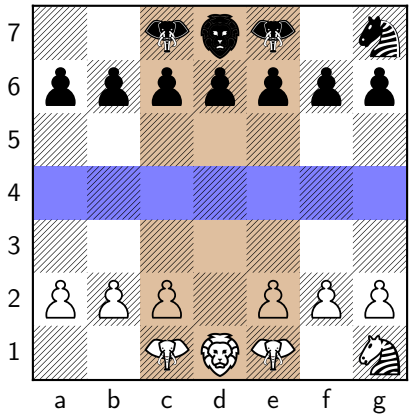

Figure 1: In the first test case, White has six pawns, two elephants and one zebra for a total score of 1300, while Black has one extra pawn, giving them 1400. The raw score is therefore -100 and since it is White to move, the evaluation remains at that value.

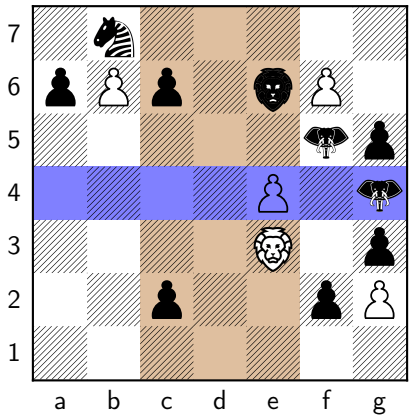

Figure 2: In this test case, White has only four pawns for a total score of 400, while Black has six pawns, two elephants and a zebra, giving them 1300. The raw score is therefore -900 and since it is Black to move, we multiply by -1 to get 900.

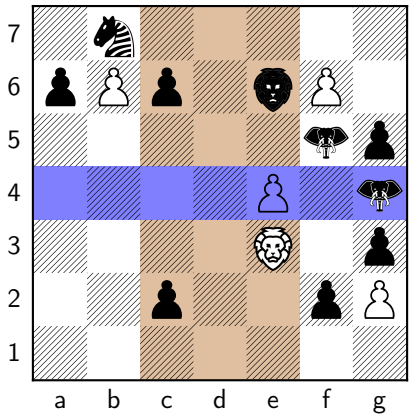

Figure 3: The same position as in the previous test case, but now it is White to play. Therefore, from White's point of view, they have a score of -900.

Submission: Search

Write a C++ program that uses the evaluation function from the previous submission and implements minimax search. The program should accept a FEN string, set up the board, and then output the minimax value when searching to a depth of 2. Pseudocode for the search algorithm is provided below:

```
function minimax(currentState, depth):
    if isGameOver(currentState) or depth <= 0:
        return evaluate(currentState)
    value = -10000000 // any large negative number
    moves = generateMoves(currentState)
    for each move in moves:
        nextState = makeMove(currentState, move)
        eval = -minimax(nextState, depth - 1)
        value = max(value, eval)
    return value
```

Input

The first line of input is N, the number of input positions given as FEN strings. N lines follow, with each line containing a single FEN string.

Output

For each FEN string, output the minimax evaluation of that position for $depth = 2$ using the aforementioned evalation function.

Example Input-Output

Sample Input

```
3
2ele1z/ppppppp/7/7/7/PPP1PPP/2ELE1Z w 4
1z5/pPp1lP1/5ep/4P1e/4L1p/2p2pP/7 b 35
1z5/pPp1lP1/5ep/4P1e/4L1p/2p2pP/7 w 12
```

Sample Output

```
-100
800
-1000
```