CS362, Homework 1: Nim

Jacob Hurst

February 4, 2019

1 Source Code

```
import sys
import itertools

#Increase this number if the maximum recursion depth is exceeded
sys.setrecursionlimit(25000)

#Used in memoizing the is_winning function
cache = {}

'''
Main entry point for the program.
'''
def nim():
    print("Welcome to the game of Nim.")

    option = input("Modes (1-5):\
                    \n\t1. determine if player can win given N and moves.\
                    \n\t2. generate a table of win statuses up to N.\
                    \n\t3. play a game of Nim against the computer.\
                    \n\t4. detect the period of winning statuses for a set
                        ↪ of moves.\
                    \n\t5. compute a subset that maximizes period - moves=
                        ↪ lower,upper (bounds on values to test).\nSelect
                        ↪ your mode: ")
```

```
    while option not in range(1,6): option = input("Invalid option, try
        ↪ again: ")

    moves = [int(move) for move in input("Provide a list of moves (comma
        ↪ separated, if single move add comma after): ")]
    moves.sort()
    n = input("Provide a starting pile size: ")

    if (option == 1): print("isWinning(" + str(n) +", " + str(moves) +")="
        ↪  + str(is_winning(n, moves)) + ".")
    elif(option == 2):
        is_winning(n, moves)
        print(cache.values())
    elif(option == 3): play(n, moves)
    elif(option == 4): print(periodicity(moves)[1])
    elif(option == 5): print(maximize_period(moves))

'''
Memoized recursive function to calculate winning positions given a
    ↪ pilesize n and a set of moves.
'''
def is_winning(n, moves):
    if(n < 0): return True
    if n not in cache:
        if(n == 0):
            cache[n] = False
            return False

        for move in moves:
            if(not is_winning(n - move, moves)):
                cache[n] = True
                return True

        cache[n] = False
        return False
    else: return cache[n]

'''
```

```
Control point for a game of nim versus the computer.
'''
def play(n, moves):
    is_winning(n, moves)
    turn = False

    while(n > 0):
        print(str(n) + " remaining in pile...")
        if(n-moves[0] < 0):
            print("Out of available moves!")
            break

        if(turn): n = human_turn(n, moves)
        else: n = computer_turn(n, moves)

        turn = not turn

    if(turn): print("Computer won!")
    else: print("Human won!")

'''
Function to process the computer players turn.
Computer player greedily takes the largest possible move that will keep
    ↪ it in a winning position.
Otherwise, it takes the smallest.
'''
def computer_turn(n, moves):
    choice = moves[0]
    for move in moves:
        index = n - move
        if(index >= 0 and not cache[index]):
            choice = move
            break

    print("Computer turn | computer took " + str(choice) + " from pile!")
    return n - choice

'''
```

```
Function to process and check the human players turn.
'''
def human_turn(n, moves):
    move = input("Human turn | provide your move: ")
    while move not in moves or n-move < 0: move = input("Invalid move, try
        ↪  again: ")
    print("Human took " + str(move) + " from pile!")
    return n - move

'''
Function to detect periodicity for a given set of moves.
Checks if the moves are strictly increasing from 1 by 1 to N. If so,
    ↪ returns the period as N+1.
Otherwise, observes two slices from the is_winning cache and searches for
    ↪  the (lcs) longest common substring
among the two. Once an lcs has been found, the function checks for
    ↪ additional repeating patterns within
the lcs. If any are found, it marks this as the pattern and updates the
    ↪ period.
'''
def periodicity(moves):
    if (increasingBy1(moves)):
        n = moves[len(moves) - 1] + 1
        is_winning(n, moves)
        values = list(cache.values())
        period = values[:n]
        return (len(period), "The period is " + str(len(period)) + ". The
            ↪ pattern is " + toString(period) + ".")

    moves.reverse()
    for move in moves:
        guess = (10 * move) + min(moves)
        for i in range(1, guess):
            n = i * guess
            is_winning(n, moves)
            values = list(cache.values())

            first = toString(values[(i - 2) * guess:(i - 1) * guess])
```

```
            second = toString(values[(i - 1) * guess:i * guess])

            lcs = longest_common_substring(first, second)
            period = principal_period(lcs)

            if period is None: period = principal_period(lcs[1:])
            if period is None: period = principal_period(lcs[:1])
            if period is None: period = double_check(first, second)
            if period is not None and len(period) != 1: return (len(period), "The period is " + str(len(period)) + ". The pattern is " + period + ".")

    if period is None: period = lcs
    return (len(period), "(Uncertain) The period is " + str(len(period)) + ". The pattern is " + period + ".")

'''
Helper function to check if a set of moves is strictly increasing by 1.
'''
def increasingBy1(moves):
    for i in range(1, len(moves)):
        if i != moves[i-1]: return False
    return True

'''
Helper function to convert a boolean list to a simpler T or F string
'''
def toString(values):
    temp = ''
    for v in values:
        if v: temp += 'T'
        else: temp += 'F'
    return temp

'''
Function to find the longest common substring among the two slices of T,F values by generating a table.
'''
```

```
def longest_common_substring(s1, s2):
   m = [[0] * (1 + len(s2)) for i in xrange(1 + len(s1))]
   longest, x_longest = 0, 0
   for x in xrange(1, 1 + len(s1)):
       for y in xrange(1, 1 + len(s2)):
           if s1[x - 1] == s2[y - 1]:
               m[x][y] = m[x - 1][y - 1] + 1
               if m[x][y] > longest:
                   longest = m[x][y]
                   x_longest = x
           else:
               m[x][y] = 0
   return s1[x_longest - longest: x_longest]

'''
Helper function to detect patterns within common substrings, helps
    ↪ shorten things.
Ex: TTFTTFTTF => TTF
'''
def principal_period(s):
    i = (s+s).find(s, 1, -1)
    return None if i == -1 else s[:i]

'''
Double checks if either of the slices are equal to eachother but just
    ↪ thrown off by 1.
Accomplishes this by sliding slices over on each by 1 and checking again.
'''
def double_check(first, second):
    if(first == second):
        first = split_check(first)
        period = first
        return period
    elif(first[1:] == second[1:]):
        first = split_check(first)
        period = first
        return period
    elif(first[1:] == second[:-1]):
```

```
        first = split_check(first)
        period = first
        return period
    elif(first[:-1] == second[1:]):
        first = split_check(first)
        period = first
        return period
    elif(first[:-1] == second[:-1]):
        first = split_check(first)
        period = first
        return period
    else: return None

'''
Helper function for double check, checks if the slice can be split
    ↪ further.
'''
def split_check(values):
    if(len(values) % 2 == 0):
        half = len(values) / 2
        vsplit1 = values[:half]
        vsplit2 = values[half:]
        if(vsplit1 == vsplit2): return vsplit1
    return values

'''
Function to detect to find a set of moves within a range of values 1..N
    ↪ with maximal period.
Sort of brute force approach, I only used on CARC with smaller values
    ↪ than {1..64}
in an attempt to detect a pattern in move sets. Works by trying m-
    ↪ combinations on periodicity function.

Maximal period nim found with form s = {1, k, k + 1}
Where k is upper bound on moves possible moves set.
Period = { 2k if k is odd
         { 2k+1 if k is even
```

```
Ex: moves = [1,14,15] => period = 28

With given subset {1,...,64}, possible maximizer is moves = [1,63,64] =>
    ↪ period = 127
'''
def maximize_period(bounds):
    lower = bounds[0]
    upper = bounds[1]
    maximum = 0
    maxiset = []

    combinations = itertools.combinations(range(upper, lower-1, -1), 3)
    for moves in combinations:
        value = periodicity(list(moves))[0]
        if(value > maximum):
            maximum = value
            maxiset = list(moves)
            print("moves="+str(maxiset)+", period="+str(maximum))

    return (maxiset, maximum)

nim()
```

2 README.md

```
<snippet>
  <content><![CDATA[
# ${1:Nim}
The game of Nim is a subtraction based game in which players take turns
    ↪ removing m marbles from a pile of size n until a player cannot make
    ↪  any further subtractions from the pile.
The last player to remove marbles wins. An initial pilesize n and a set
    ↪ of valid moves m (1 < m <= N) are needed to play the game.

Typically, the game is played with more than one pile. This program is a
    ↪ variant one pile game with additional modes described below.
```

```
## Installation
To get started, download Nim.py and run 'python Nim.py'.
## Usage
There are 5 modes the user can choose from - detailed below:
1. Detect position winning status, returns True if player can win given
    ↪ pilesize N and the moveset
2. Generate a table of winning statuses from 0 up to pilesize N for the
    ↪ given moveset.
3. Play a game of Nim against the computer with the given pilesize N and
    ↪ the moveset.
4. Detect the periodicity of winning statuses for a set of moves and
    ↪ arbitrary pilesizes N.
5. Compute a subset of moves = (1, N) that maximizes period - input:
    ↪ moves as lower & upper bounds.
]]></content>
  <tabTrigger>readme</tabTrigger>
</snippet>
```

3 Sample IO Transcripts

```
Sample Input / Output

Part 1a:

Welcome to the game of Nim.
Modes (1-5):
        1. determine if player can win given N and moves.
        2. generate a table of win statuses up to N.
        3. play a game of Nim against the computer.
        4. detect the period of winning statuses for a set of moves.
        5. compute a subset that maximizes period - moves=lower,upper (
            ↪ bounds on values to test).
Select your mode: 1
Provide a list of moves (comma separated, if single move add comma after)
    ↪ : 1,2,3
Provide a starting pile size: 100
```

```
isWinning(100, [1, 2, 3])=False.

Welcome to the game of Nim.
Modes (1-5):
        1. determine if player can win given N and moves.
        2. generate a table of win statuses up to N.
        3. play a game of Nim against the computer.
        4. detect the period of winning statuses for a set of moves.
        5. compute a subset that maximizes period - moves=lower,upper (
            ↪ bounds on values to test).
Select your mode: 1
Provide a list of moves (comma separated, if single move add comma after)
    ↪ : 1,4,5,10
Provide a starting pile size: 1000
isWinning(1000, [1, 4, 5, 10])=True.

Welcome to the game of Nim.
Modes (1-5):
        1. determine if player can win given N and moves.
        2. generate a table of win statuses up to N.
        3. play a game of Nim against the computer.
        4. detect the period of winning statuses for a set of moves.
        5. compute a subset that maximizes period - moves=lower,upper (
            ↪ bounds on values to test).
Select your mode: 1
Provide a list of moves (comma separated, if single move add comma after)
    ↪ : 1,2,3,4,5,6
Provide a starting pile size: 14000
isWinning(14000, [1, 2, 3, 4, 5, 6])=False.

Part 1b:

Welcome to the game of Nim.
Modes (1-5):
        1. determine if player can win given N and moves.
        2. generate a table of win statuses up to N.
        3. play a game of Nim against the computer.
        4. detect the period of winning statuses for a set of moves.
```

```
        5. compute a subset that maximizes period - moves=lower,upper (
           ↪ bounds on values to test).
Select your mode: 3
Provide a list of moves (comma separated, if single move add comma after)
   ↪ : 1,2,3,4,5,6,7,8,9
Provide a starting pile size: 45
45 remaining in pile...
Computer turn | computer took 5 from pile!
40 remaining in pile...
Human turn | provide your move: 4
Human took 4 from pile!
36 remaining in pile...
Computer turn | computer took 6 from pile!
30 remaining in pile...
Human turn | provide your move: 7
Human took 7 from pile!
23 remaining in pile...
Computer turn | computer took 3 from pile!
20 remaining in pile...
Human turn | provide your move: 7
Human took 7 from pile!
13 remaining in pile...
Computer turn | computer took 3 from pile!
10 remaining in pile...
Human turn | provide your move: 2
Human took 2 from pile!
8 remaining in pile...
Computer turn | computer took 8 from pile!
Computer won!

Part 2a:

Welcome to the game of Nim.
Modes (1-5):
        1. determine if player can win given N and moves.
        2. generate a table of win statuses up to N.
        3. play a game of Nim against the computer.
        4. detect the period of winning statuses for a set of moves.
```

```
        5. compute a subset that maximizes period - moves=lower,upper (
            ↪ bounds on values to test).
Select your mode: 4
Provide a list of moves (comma separated, if single move add comma after)
    ↪ : 1,2,3
Provide a starting pile size: 1
The period is 4. The pattern is FTTT.

Welcome to the game of Nim.
Modes (1-5):
        1. determine if player can win given N and moves.
        2. generate a table of win statuses up to N.
        3. play a game of Nim against the computer.
        4. detect the period of winning statuses for a set of moves.
        5. compute a subset that maximizes period - moves=lower,upper (
            ↪ bounds on values to test).
Select your mode: 4
Provide a list of moves (comma separated, if single move add comma after)
    ↪ : 1,4,5,10
Provide a starting pile size: 1
The period is 3. The pattern is TFT.
***(Wasn't sure why but it was identifying the pattern shifted by 1. This
    ↪  persists in later pattern findings also.)

Welcome to the game of Nim.
Modes (1-5):
        1. determine if player can win given N and moves.
        2. generate a table of win statuses up to N.
        3. play a game of Nim against the computer.
        4. detect the period of winning statuses for a set of moves.
        5. compute a subset that maximizes period - moves=lower,upper (
            ↪ bounds on values to test).
Select your mode: 4
Provide a list of moves (comma separated, if single move add comma after)
    ↪ : 1,2,4,5,7,8,9,10
Provide a starting pile size: 1
The period is 101. The pattern is
    ↪ TFTTTTTTTTTTFTTFTTFTTTTTTTTTTFTTFTTFTTTTTTTTTTFTTFTTFTTTTTTTTTT
```

```
FTTFTTFTTTTTTTTTTFTTFTTFTTTTTTTTTTFTTF.

Welcome to the game of Nim.
Modes (1-5):
        1. determine if player can win given N and moves.
        2. generate a table of win statuses up to N.
        3. play a game of Nim against the computer.
        4. detect the period of winning statuses for a set of moves.
        5. compute a subset that maximizes period - moves=lower,upper (
            ↪ bounds on values to test).
Select your mode: 4
Provide a list of moves (comma separated, if single move add comma after)
    ↪ : 4,5
Provide a starting pile size: 1
The period is 44. The pattern is
    ↪ FFFFTTTTTFFFFTTTTTFFFFTTTTTFFFFTTTTTFFFFTTTT.

Part 3 (my own sample inputs):

Welcome to the game of Nim.
Modes (1-5):
        1. determine if player can win given N and moves.
        2. generate a table of win statuses up to N.
        3. play a game of Nim against the computer.
        4. detect the period of winning statuses for a set of moves.
        5. compute a subset that maximizes period - moves=lower,upper (
            ↪ bounds on values to test).
Select your mode: 4
Provide a list of moves (comma separated, if single move add comma after)
    ↪ : 1,14,15
Provide a starting pile size: 1
The period is 28. The pattern is FTFTFTFTFTFTTTTTTTTTTTTTTTFT.

***This is also my submission for the subset that maximizes period.
   I noticed this pattern by a function that tried combinations on a
      ↪ smaller set {1..16}.
   It seems to be of the form {1, k, k+1} => period = 2k or 2k+1
Welcome to the game of Nim.
```

```
Modes (1-5):
        1. determine if player can win given N and moves.
        2. generate a table of win statuses up to N.
        3. play a game of Nim against the computer.
        4. detect the period of winning statuses for a set of moves.
        5. compute a subset that maximizes period - moves=lower,upper (
            ↪ bounds on values to test).
Select your mode: 4
Provide a list of moves (comma separated, if single move add comma after)
    ↪ : 1,63,64
Provide a starting pile size: 1
The period is 127. The pattern is
    ↪ FTFTFTFTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
TTTTTTFTFTFTFTFTFTFTFTFTFTFTFTFTFTFTFTFTFTFTFTFTFTFTFTFTFTFTFT.

Welcome to the game of Nim.
Modes (1-5):
        1. determine if player can win given N and moves.
        2. generate a table of win statuses up to N.
        3. play a game of Nim against the computer.
        4. detect the period of winning statuses for a set of moves.
        5. compute a subset that maximizes period - moves=lower,upper (
            ↪ bounds on values to test).
Select your mode: 2
Provide a list of moves (comma separated, if single move add comma after)
    ↪ : 1,4,5,10
Provide a starting pile size: 20
[False, True, False, True, True, True, True, True, False, True, True,
    ↪ False, True, True, False, True, True, False, True, True, False]
```