Machine Learning in Practice

#7: Combinatorial Games: Programming Exercise

Summer 2019

```
def AcanWin (state):
    if the game ends in this state with B winning:
      return False

    for each of the A's possible move:
        state' = new state reached by the move
        if not BcanWin (state'):
            return True  # state => state' guarantees a win for A

    return False  # whichever move A selects, B wins

def BcanWin (state):
    if the game ends in this state with A winning:
      return False

    for each of the B's possible move:
        state' = new state reached by the move
        if not AcanWin (state'):
            return True  # state => state' guarantees a win for B

    return False  # whichever move B selects, A wins

# A is the winner if AcanWin(starting_state) is True.
# Otherwise, B is the winner
```

For impartial games, `AcanWin` and `BcanWin` can be merged into one function `canWin`.

```
def canWin (state):
    if the game ends in this state with the other player winning:
      return False

    for each of this player's possible move:
        state' = new state reached by the move
        if not canWin (state'):
            return True

    return False
```

1. (P1.py)

Consider the following partizan game which is similar to the Nim game:

- There are n stones in a pile.
- Two players A and B take turns, where player A takes first, and remove some specified number of stones at a time:
  - player A can remove 1, 2, 3, 4, 5, or 6 stones at a time.
  - player B can remove 2, 3, 5, or 7 stones at a time.
- When the number of remaining stones is smaller than a number of stones that can be removed by a player at a time, all the remaining stones can be removed by the player.
  - e.g. if 4 stones remains at B's turn, B can remove all the 4 stones (since B can remove 5 or 7 stones at a time).
- The one who takes the last stone wins.
- Who can force a win (assuming both players play perfectly)?

Write a function `AcanWin` (along with `BcanWin`) that determines whether or not the first player can force a win from a given starting configuration.

- input parameter: an integer `n` that represents the initial number of stones
- return value:
  - `True` if player A can force a win
  - `False` if player B can force a win

2. (P2.py)

Players A and B play the following impartial game:

- Two positive integers n and k are given by the dealer of the game.
- Player A starts the game by selecting an integer $a_1 \in \{2, 3, \ldots, 9\}$. Let $x = k * a_1$.
- Player B selects an integer $b_1 \in \{2, 3, \ldots, 9\}$. Let $x = x * b_1$.
- Player A selects an integer $a_2 \in \{2, 3, \ldots, 9\}$. Let $x = x * a_2$.
- Player B selects an integer $b_2 \in \{2, 3, \ldots, 9\}$. Let $x = x * b_2$.
- ...
- Whoever first makes $x \geq n$ wins the game.

Assuming that both players are sufficiently clever and do their best, exactly one of the players can force a win (for each fixed (n, k)).

Write a function `canWin`:

- input parameter: two integers k and n where $k < n$
- return value:
  - `True`: if player A can force a win in the game defined by n and k
  - `False`: otherwise

3. (P3.py)

Consider the following impartial game played by two players, A and B:

- Initially, a positive integer $n > 5$ is known to both players.
- A begins the game by selecting $n_1 = 1$.
- B responds by selecting $n_2 \in \{n_1+1,\ 2n_1+2,\ n_1^2+1\}$ $(= \{2, 4\})$.
- $\cdots$
- A and B take turns by selecting $n_{i+1} \in \{n_i+1,\ 2n_i+2,\ n_i^2+1\} \cap \{1, 2, \cdots, n\}$.
  - i.e. players cannot select a number larger than n
- The one who first selects n wins.
- Who can force a win?

Write a function `canWin` that determines whether or not the first player A can force a win for each n.

- input parameter: an integer $n > 5$
- return value:
  - `True`: if player A can force a win for the given n
  - `False`: if player B can force a win for the given n

4. (P4.py)

포로리와 아로리는 (impartial) 카드 게임을 하고 있다. n개의 카드가 일렬로 놓여 있다. 각 카드에는 점수가 적혀있다. 포로리부터 시작하여 번갈아가면서 턴이 진행되는데 한 턴에는 가장 왼쪽에 있는 카드나 가장 오른쪽에 있는 카드를 가져갈 수 있다. 카드가 더 이상 남아있지 않을 때까지 턴은 반복된다. 게임의 점수는 자신이 가져간 카드에 적힌 수의 합이다.

포로리와 아로리는 서로 자신의 점수를 가장 높이기 위해 최선의 전략으로 게임에 임한다. 예를 들어 카드가 $[4,3,1,2]$로 놓여있을 때 각자의 최선의 전략은 다음과 같다: 포로리는 처음에 4가 적힌 카드를 가져가고, 아로리는 3이 적힌 카드를 가져간다. 그리고 포로리는 2가 적힌 카드를 가져가고, 아로리는 마지막으로 1이 적힌 카드를 가져간다. 이 때 포로리가 얻는 점수는 6이다.

다음과 같은 입력과 출력을 가지는 함수 `f`를 작성하라:

- 입력: 카드에 써져 있는 숫자들의 list `L`
- 출력: 포로리와 아로리가 최선의 전략으로 임하면서 포로리가 `L`로 게임을 시작할 때 포로리가 얻게 되는 점수
  - 포로리와 아로리가 최선의 전략으로 임하면서 아로리가 `L`로 게임을 시작할 때 아로리가 얻게 되는 점수도 이 값과 같다 (impartial)

힌트:

- $\texttt{f(L)} = \max \begin{cases} \texttt{L[0]} + \big(\texttt{sum(L[1:])} - \texttt{f(L[1:])}\big) & \text{왼쪽을 택할 경우} \\ \texttt{L[n-1]} + \big(\texttt{sum(L[:n-1])} - \texttt{f(L[:n-1])}\big) & \text{오른쪽을 택할 경우} \end{cases}$

위의 exponential-time recursive 방식외에 `P4_DP.py`와 같이 dynamic programming 기반 $O(N^2)$ 알고리즘으로 구현할 수도 있다.