

# Computer Implementation of Multiple Fibonacci Nim

*Chizhong Zhou*  
chizhongz@163.com

Department of Computer and Information Engineering  
Hunan Institute of Science & Technology, Yueyang, Hunan 414000, PR China

**Abstract:** In this paper a special combinatorial game with two players, called multiple Fibonacci Nim, is defined as below:

There are  $n > 1$  piles of chips. The number of chips in each pile is greater than 1. Two players take chips alternately. Each player in his turn first chooses one of the piles, and he can change his choice of piles in any his other turn. He then removes chips from the pile. The first player may remove any number of chips from the pile he chosen but not the whole pile. After that each player may remove at most twice as many as the previous player removed. The winner is the player who removes the last chip.

Properties of general multiple Fibonacci Nim are researched. For  $n = 2$  and  $n = 3$ , the algorithms or part of source codes to determine the  $\mathcal{P}$  positions and to find the winning strategy are given.

## 1. Introduction

The Fibonacci Nim presented by Whinihan, [9],[1], is a take-away game where the players alternately remove a positive number of chips from a single pile, the player removing the last counter being the winner. On the initial move, the first player may remove any number of chips, so long as he leaves at least one. On each subsequent move, a player may remove at most  $2a$  chips, where  $a$  is the number of chips removed by his opponent on the preceding move.

There are two directions to generalize the game. In one direction the restriction  $2a$  is generalized to a function  $f(a)$ . A number of articles in the direction have been published (e.g.[2],[5],[6],[7],[8]). In another direction, maybe being more difficult, the condition of single pile is generalized to the condition of more piles. There are few articles devoted to the last direction. In [3] and [4] games with two piles and with generalized restriction have been studied. In this paper we deal with another Fibonacci Nim with more piles, called multiple Fibonacci Nim. We define it as below:

There are  $n > 1$  piles of chips. The number of chips in each pile is greater than 1. Two players take chips alternately. Each player in his turn first chooses one of the piles, and he can change his choice of piles in any his other turn. He then removes chips from the pile. The first player may remove any number of chips from the pile he chosen but not the whole pile. After that each player may remove at most twice as many as the previous player removed. The winner is the player who removes the last chip.

It can be observed that the games studied in [3] and [4] are not the multiple Fibonacci Nim since the condition for game to end is one of the two piles is empty. For the Fibonacci Nim, it is well known that if the initial number of chips is a Fibonacci number then it's a  $\mathcal{P}$  position (Previous player winning [1]), otherwise it's a  $\mathcal{N}$  position (Next player winning [1]). And the

Zeckendorf theorem is well applied to find the winning strategy. However, for the multiple Fibonacci Nim it's difficult to get a rule for determining whether a position is  $\mathcal{P}$  position. So we try to use a computer to find the  $\mathcal{P}$  positions and to implement the winning strategy for the multiple Fibonacci Nim. Yet it is also difficult. We call  $\xi = (a_1, \dots, a_n; x) = (a_{i_1}, \dots, a_{i_n}; x)$  a position, where  $a_1, \dots, a_n$  are the numbers of chips in each of the  $n$  piles,  $x$  is the number of chips the previous player removed (For the initial position  $x = 0$ ), and  $a_{i_1}, \dots, a_{i_n}$  is any permutation of  $a_1, \dots, a_n$ . Suppose that the number of chips in each pile is at most  $N$ . Then there are at most  $N^{n+1}/n!$  different positions. It needs too more memories if we want to store the statuses ( $\mathcal{P}$  position or  $\mathcal{N}$  position) of these positions. In Section 2 we discuss several properties of general multiple Fibonacci Nim. In Section 3 we discuss special properties of two pile multiple Fibonacci Nim and its implementation. In Section 4 we discuss the implementation of 3 pile multiple Fibonacci Nim.

## 2. Several properties of general multiple Fibonacci Nim

We denote the set of  $\mathcal{P}$  positions by  $\mathcal{P}$  and the set of  $\mathcal{N}$  positions by  $\mathcal{N}$ . Since  $(0, \dots, 0; x)$ ,  $x > 0$ , is a terminal position, therefore

**Theorem 2.1.** For  $x > 0$ ,  $(0, \dots, 0; x) \in \mathcal{P}$ .

**Theorem 2.2.** Let  $x > 0$ , and let  $a_1, \dots, a_n$  be not all zero. If  $\xi = (a_1, \dots, a_n; x) \in \mathcal{P}$ , then for any  $0 < y \leq x$ ,  $\eta = (a_1, \dots, a_n; y) \in \mathcal{P}$ .

*Proof.* For any  $a_i \neq 0$  ( $1 \leq i \leq n$ ),  $\xi = (a_1, \dots, a_n; x)$  has followers  $\xi_{i,j} = (a_1, \dots, a_i - j, \dots, a_n; j)$ ,  $1 \leq j \leq \min(a_i, 2x)$ , and  $\eta = (a_1, \dots, a_n; y)$  has followers  $\eta_{i,j} = (a_1, \dots, a_i - j, \dots, a_n; j)$ ,  $1 \leq j \leq \min(a_i, 2y)$ . Since  $0 < y \leq x$  the set of followers of  $\eta$  is included in the set of followers of  $\xi$ . The fact that  $\xi$  is a  $\mathcal{P}$  position implies that its all followers are  $\mathcal{N}$  positions. Whence all followers of  $\eta$  are  $\mathcal{N}$  positions. So  $\eta$  is a  $\mathcal{P}$  position.  $\square$

**Corollary 2.3.** Let  $y > 0$ , and let  $a_1, \dots, a_n$  be not all zero. If  $\eta = (a_1, \dots, a_n; y) \in \mathcal{N}$ , then for any  $x \geq y$ ,  $\xi = (a_1, \dots, a_n; x) \in \mathcal{N}$ .

**Theorem 2.4.** Let  $x > 0$ , and let  $a_1, \dots, a_n$  be not all zero. If  $\xi = (a_1, \dots, a_n; x) \in \mathcal{P}$ , then for any  $1 \leq i \leq n$ ,  $0 < j \leq x$  and  $2y \geq j$ ,  $\xi_{i,j,y} = (a_1, \dots, a_i + j, \dots, a_n; y) \in \mathcal{N}$ .

*Proof.* From  $2y \geq j$  we observe that  $\eta = (a_1, \dots, a_n; j)$  is a follower of  $\xi_{i,j,y}$ . By theorem 2.2,  $\eta \in \mathcal{P}$  since  $\xi \in \mathcal{P}$ . Hence  $\xi_{i,j,y} \in \mathcal{N}$ .  $\square$

**Corollary 2.5.** Let  $x > 0$ , and let  $a_1, \dots, a_n$  be not all zero. Assume  $\xi = (a_1, \dots, a_n; x) \in \mathcal{P}$ . If there exist  $1 \leq i \leq n$ ,  $j > 0$  and  $y > 0$  such that  $\xi_{i,j,y} = (a_1, \dots, a_i + j, \dots, a_n; y) \in \mathcal{P}$ , then  $j > x$  or  $j > 2y$ . Specifically, under above conditions, if  $(a_1, \dots, a_n; 1) \in \mathcal{P}$  and  $(a_1, \dots, a_i + j, \dots, a_n; 1) \in \mathcal{P}$  then  $j \geq 2$ .

**Theorem 2.6.** Let  $\xi_0 = (a_1, \dots, a_n; 0)$ , where  $a_1, \dots, a_n$  are all greater than 1.

- (1) If  $a_1, \dots, a_n$  are all greater than 2 and  $\xi_1 = (a_1, \dots, a_n; 1) \in \mathcal{N}$  then  $\xi_0 \in \mathcal{N}$ ;
- (2) If  $\xi_1 \in \mathcal{P}$  and denote  $x^* = \max\{x \mid x > 0 \text{ and } (a_1, \dots, a_n; x) \in \mathcal{P}\}$ . Then  $\xi_0 \in \mathcal{P}$  iff for any  $1 \leq i \leq n$ ,  $a_i > 2x^* + 1$  implies that  $(a_1, \dots, a_i - 2x^* - 1, \dots, a_n; 2x^* + 1), \dots, (a_1, \dots, 1, \dots, a_n; a_i - 1)$  are all  $\mathcal{N}$  positions.

*Proof.* For any  $a_i$ ,  $\xi_0$  has followers  $\xi_{i,j} = (a_1, \dots, a_i - j, \dots, a_n; j)$ ,  $j = 1, \dots, a_i - 1$ .

(1) We observe that the set of followers of  $\xi_0$  includes the set of followers of  $\xi_1$  since  $a_1, \dots, a_n$  are all greater than 2. If  $\xi_1 \in \mathcal{N}$  then it has a follower of  $\mathcal{P}$  position and so  $\xi_0$  has a follower of  $\mathcal{P}$  position. Thus  $\xi_0 \in \mathcal{N}$ ;

(2) If  $\xi_1 \in \mathcal{P}$  then  $x^*$  exists. For any  $a_i$ ,  $\xi^* = (a_1, \dots, a_n; x^*)$  has followers  $\xi_{i,j}^* = (a_1, \dots, a_i - j, \dots, a_n; j)$ ,  $j = 1, \dots, \min(a_i, 2x^*)$ . They are all  $\mathcal{N}$  positions. Whence, if  $a_i \leq 2x^* + 1$  then for  $j = 1, \dots, \min(a_i - 1, 2x^*)$ ,  $\xi_{i,j} = \xi_{i,j}^*$  are all  $\mathcal{N}$  positions. However, if  $a_i > 2x^* + 1$ , then  $\eta_{i,2x^*+1}^* = (a_1, \dots, a_i - 2x^* - 1, \dots, a_n; 2x^* + 1), \dots, \eta_{i,a_i-1}^* = (a_1, \dots, 1, \dots, a_n; a_i - 1)$  are not included in these  $\xi_{i,j}^*$ 's. So, if  $\xi_0 \in \mathcal{P}$  it must be  $\eta_{i,j}^* \in \mathcal{N}$ ,  $j = 2x^* + 1, \dots, a_i - 1$ . The necessity is proved. The sufficiency is obviously.  $\square$

**Theorem 2.7.** *Let  $a_1 \geq a_2 \geq \dots \geq a_n \geq 0$  and  $a_1, \dots, a_n$  be not all zero. Suppose  $2x \geq a_1$ . Then for any  $y \geq x$ ,  $(a_1, \dots, a_n; y) \in \mathcal{P}$  iff  $(a_1, \dots, a_n; x) \in \mathcal{P}$ .*

*Proof.* Since  $2y \geq 2x \geq a_1 = \max(a_1, \dots, a_n)$ , therefor the set of followers of  $(a_1, \dots, a_n; y)$  is equal to the set of followers of  $(a_1, \dots, a_n; x)$ . And the proof is finished.  $\square$

### 3. Two pile Fibonacci Nim

#### 3.1. Properties and algorithm of two pile Fibonacci Nim

The two pile Fibonacci Nim has very good properties so that we can easily solve it.

**Theorem 3.1.** *Let  $m \geq 0$  and  $x > 0$ . Then*

- (1)  $\xi = (m, m; x) \in \mathcal{P}$ ;
- (2) For  $0 < k \leq 2x$ ,  $\eta = (m + k, m; x) \in \mathcal{N}$ .

*Proof.* We prove the theorem by induction. For  $m = 0$ ,  $\xi$  is a terminal position, and  $\eta$  has a follower of terminal position  $(0, 0; k)$ . Whence the conclusions hold. Suppose that the conclusions hold for  $0 \leq m \leq q$ . We want to prove  $\xi' = (q + 1, q + 1; x) \in \mathcal{P}$  and, for  $0 < k \leq 2x$ ,  $\eta' = (q + 1 + k, q + 1; x) \in \mathcal{N}$ . Any follower of  $\xi'$  has the form  $\xi_j' = (q + 1, q + 1 - j; j) = (m_1 + k_1, m_1; j)$ , where  $m_1 = q + 1 - j$ ,  $k_1 = j$  and  $1 \leq j \leq \min(q + 1, 2x)$ . Since  $0 \leq m_1 \leq q$  and  $0 < k_1 \leq 2x$ , therefor by the induction hypothesis  $\xi_j' \in \mathcal{N}$ . This implies  $\xi \in \mathcal{P}$ . Hence for all  $m \geq 0$  the conclusion (1) holds. Now  $\eta'$  has a follower  $\eta_j' = (q + 1, q + 1; k)$  which, by the result just proved, is a  $\mathcal{P}$  position. Whence  $\eta' \in \mathcal{N}$ . And so the proof is completed.  $\square$

**Corollary 3.2.** *For  $m > 1$  and  $k > 0$ ,  $\eta = (m + k, m; 0) \in \mathcal{N}$ .*

*Proof.* It follows from that  $\eta$  has a follower  $(m, m; k) \in \mathcal{P}$ .  $\square$

By using Theorem 3.1 and its corollary we can get the algorithm for winning strategy. The algorithm is given only for the initial  $\mathcal{N}$  position  $(m + k, m; 0)$ , where, as known, the player first to move has a winning strategy. In the later discussions we call the player first to move Left and the player next to move Right. Denote the number of chips Left removed by L and that Right removed by R.

**Algorithm Mult2FibGame:**

- |                                         |                                                                      |
|-----------------------------------------|----------------------------------------------------------------------|
| (1) Input $m \geq 0$ and $k > 0$ .      | (4) Input $R$ (Must be $0 < R \leq \min(m, 2L)$ ), $m=m-R$ , $k=R$ . |
| (2) $L=k$ from the pile of size $m+k$ . | (5) Go to (2).                                                       |
| (3) If $m=0$ go to (6).                 | (6) End.                                                             |

**3.2.  $\mathcal{P}$  position and  $\mathcal{N}$  position in single pile game and two pile game**

There is another problem. At step (2) of Algorithm Mult2FibGame if the Left changes his move a position of the form  $\eta = (m + k, m; x)$  with  $k > 2x$  may appear which is not of the forms in Theorem 3.1 and its corollary. For example, let the initial position be  $(12, 5; 0)$ . The Left remove 1 chip from the pile of size 5 to get position  $(12, 4; 1)$ . Here  $m = 4$ ,  $k = 8$ ,  $x = 1$ , and so  $k > 2x$ . Under this position whether the Left can win yet? For answering the problem and for the requirement in the next section we analyze the conditions for this position to be a  $\mathcal{P}$  position. Any follower of  $\eta = (m + k, m; x)$ ,  $k > 2x$ , may be in one of the 3 forms:

*Form1.*  $\{\tau_{i,1} = (m + k, m - i; i)\}$ ,  $1 \leq i \leq \min(m - 1, 2x)$ ;

*Form2.*  $\{\tau_{i,2} = (m + k, 0; m)\}$ ,  $i = m \leq 2x$ ;

*Form3.*  $\{\tau_{i,3} = (m + k - i, m; i)\}$ ,  $1 \leq i \leq 2x$ .

A position of *Form2* is essentially a position in single pile game. To be simple we denote the position  $(m, 0; x)$  by form  $(m; x)$ . To determine whether it is a  $\mathcal{P}$  position we have the following:

**Lemma 3.3.** *Let  $m > 0$  and  $x > 0$ . If  $m \leq 2x$  then  $(m; x) \in \mathcal{N}$ .*

*Proof.* It follows from that  $m \leq 2x$ ,  $(m; x)$  has the terminal position  $(0; m)$  as its follower.  $\square$

**Theorem 3.4.** *Let  $m > 0$  and  $x > 0$ . If  $m > 2x$  then  $(m; x) \in \mathcal{N}$  iff there exists an  $i$ ,  $1 \leq i \leq 2x$  and  $3i < m$ , such that  $(m - i; i) \in \mathcal{P}$ .*

*Proof.* Any follower of  $(m; x)$  has the form  $(m - i; i)$  with  $1 \leq i \leq 2x$ . If  $(m; x) \in \mathcal{N}$  one of its follower  $(m - i; i)$  must be  $\mathcal{P}$  position. From Lemma 3.3 it must be  $m - i > 2i$  which is  $3i < m$ . Conversely, if for all  $1 \leq i \leq 2x$  and  $3i < m$ ,  $(m - i; i) \in \mathcal{N}$ , then all followers of  $(m; x)$  are  $\mathcal{N}$  position, and so  $(m; x) \in \mathcal{P}$ . This contradicts the assumption.  $\square$

A position of *Form1* has the same form as  $\eta$ . While a position of *Form3* has the same form as  $\eta$  only for  $k > 3i$ . Hence we have

**Theorem 3.5.** *Let  $m > 0$ ,  $k > 0$ ,  $x > 0$  and  $k > 2x$ . Then  $\eta = (m + k, m; x) \in \mathcal{N}$  iff there exists an  $i$ ,  $1 \leq i \leq 2x$ , such that  $i < m$  and  $(m + k, m - i; i) \in \mathcal{P}$  or  $i = m$  and  $(m + k; i) \in \mathcal{P}$  or  $3 * i < k$  and  $(m + k - i, m; i) \in \mathcal{P}$ .*

*Proof.* From the above state and Theorem 3.1 the proof is similar to that of Theorem 3.4.  $\square$

Now we discuss the algorithms for determining  $(m; x) \in \mathcal{P}$  and  $(m + k, m; x) \in \mathcal{P}$ ,  $k > 2x$ . For the former, define a function  $y = IsPPos1(m; x)$ :  $y = 1$  If  $(m; x) \in \mathcal{P}$ , and  $y = 0$  otherwise. From Theorem 3.4 we can compute the value of  $IsPPos1(m; x)$  recursively. Here we give an algorithm to compute  $IsPPos1(m; x)$  from  $m = 1$  to a given positive integer  $N$ .

**Algorithm PPos1:**

- |                                           |                                      |
|-------------------------------------------|--------------------------------------|
| (1) $m=3$ .                               | (8) $i=i+1$ , go to (6).             |
| (2) If $m>N$ go to (13).                  | (9) $IsPPos1(m, x)=1$ , go to (11).  |
| (3) $x=1$ .                               | (10) $IsPPos1(m, x)=0$ , go to (12). |
| (4) If $2x \geq m$ go to (12).            | (11) $x=x+1$ , go to (4).            |
| (5) $i=1$ .                               | (12) $m=m+1$ .                       |
| (6) If $3i \geq m$ or $i > 2x$ go to (9). | (13) End.                            |
| (7) If $IsPPos1(m-i, i)=1$ go to (10).    |                                      |

To implement the algorithm the values of the function computed must be stored. For saving the memory and the time we only store those positions  $(m; x)$ 's that  $isPpos(m; x) = 1$ . We adapt a structure array to store these positions defined as below:

```
#define N 51
/* The member prev is the number of chips removed by the previous player. */
typedef struct node {int a, prv; }Node;
Node PPos1Arr[(N+1)/2];
```

That the length of the array is  $(N + 1)/2$  follows from Corollary 2.5. If  $isPpos(m; x)=1$  we store  $(m; x)$  in PPos1Arr. Furthermore, by Theorem 2.2, if  $(m; 1), \dots, (m; x)$  are all  $\mathcal{P}$  positions but  $(m; x + 1)$  is not, we need only to store  $(m; x)$ . And the function  $IsPPos1(m; x)$  is replaced by the function  $InPpos1(m, x)$  which returns 1 if  $(m; x)$  is in PPos1Arr, and returns 0 otherwise. The part of source code in C is

```
int count=0; /* Global variable for counting the elements of PPos1Arr */
m0=0; /* For  $x>0$ ,  $(0, x) \in \mathcal{P}$ . */
for(m=3; m<=N; m++) /* By Lemma 3.3, for  $x>0$ , if  $m=1$  or  $2$ , */
for(x=1; 2*x<m; x++) /* or  $m \leq 2x$  then  $(m; x) \in \mathcal{N}$ . */
{flag=1;
for(i=1; 3*i<m&&i<=2*x; i++)
if(InPPos1(m-i, i)==1) {flag=0; break;} /* By Theorem 3.4  $(m; x) \in \mathcal{N}$ . */
if(flag)
{if(m==m0) /*  $(m; x - 1)$  has been in PPos1Arr. */
{j=count-1; PPos1Arr[j].prv=x; }
else /*  $(m; x - 1)$  has not been in PPos1Arr. */
{j=count; PPos1Arr[j].a=m; PPos1Arr[j].prev=1; count++;
m0=m; /* New  $\mathcal{P}$  position  $(m; 1)$ . */
}
}
else break; /* By Corollary 2.3. */
}

int InPPos1(int m,int n)
{int i;
for(i=0; i<count; i++)
if(PPos1Arr[i].a==m&&PPos1[i].prev>=n) return 1; /* By Theorem 2.2. */
```

```

return 0;
}

```

Similarly, if  $(m+k, m; x)$ ,  $k > 2x > 0$  is a  $\mathcal{P}$  position we can store it in a structure array (To be convenient, for every  $m$  we define an array  $\text{PPos2Arr}[m]$  which is dynamic). Also we can define a function  $\text{InPPos2}(k, m, x)$  to characterize whether the position  $(k, m; x)$  has been stored in the array  $\text{PPos2Arr}[m]$ . The length of each array is also determined by Corollary 2.5. The part of source code in C is

```

Node *PPos2Arr[N];
int t[N]; /* To count the elements of each dynamic array. */
for(m=1; m<N; m++){t[m]=0;
  PPos2Arr[m]=(Node *) malloc((N-m)/2*sizeof(Node)); /* Dynamic arrays */
}
m0=0; k0=0; /* For x>0, (0, 0; x) ∈ P. */
for(m=1; m<N; m++)
  for(k=1; k<=N-m; k++)
    for(x=1; 2*x<k; x++) /* By Theorem 3.1 if k ≤ 2x then (m; x) ∈ N. */
      {flag=1;
        for(i=1; i<=2*x; i++)
          if(i<m&&InPPos2(m+k,m-i,i)||i==m&&InPPos1(m+k,i)||3*i<k&&InPPos2(m+k-i,m,i))
            {flag=0; break;} /* By Theorem 3.5 (m; x) ∈ N. */
          if(flag)
            {if(m==m0&&k==k0)/* (m+k, m; x-1) has been in PPos2Arr[m]. */
              {j=t[m]-1; PPos2Arr[m][j].prev=x; }
              else /* (m+k, m; x-1) has not been in PPos2Arr[m]. */
                {j=t[m]; PPos2Arr[m][j].a=m+k; PPos2Arr[m][j].prev=1; t[m]++;
                  m0=m; k0=k; /* New P position (m+k, m; 1). */
                }
            }
        }
      else break; /* By Corollary 2.3. */
    }

int InPPos2(int k,int m,int x)
{int i;
  for(i=0; i<t[m]; i++)
    if(k==PPos2Arr[m][i].a&&PPos2Arr[m][i].prev>=x) return 1; /* By Theorem 2.2. */
  return 0;
}

```

#### 4. Three pile Fibonacci Nim

The multiple Fibonacci game for  $n = 3$  is more complicated. It seems that we could not found its so good properties as  $n = 2$ . Any position  $(h, k, m; x)$  in this game may be in one of the 3 classes:

*C1.*  $hkm \neq 0$ ,  $x \geq 0$ ; *C2.* Just one of  $h, k, m$  is zero and  $x > 0$ ; *C3.* Two or three of  $h, k, m$  are zero and  $x > 0$ ;

**Theorem 4.1.** Let  $hkm \neq 0$ ,  $x > 0$ . Then  $(h, k, m; x) \in \mathcal{N}$  iff there exists an  $i$ ,  $1 \leq i \leq 2x$ , such that one of following conditions holds:

- (1)  $i = m$  and  $h = k$ , or  $i = m$  and  $h \neq k$  and  $(h, k; i) \in \mathcal{P}$ ;
- (2)  $i = k$  and  $h = m$ , or  $i = k$  and  $h \neq m$  and  $(h, m; i) \in \mathcal{P}$ ;
- (3)  $i = h$  and  $k = m$ , or  $i = h$  and  $k \neq m$  and  $(k, m; i) \in \mathcal{P}$ ;
- (4)  $i < m$  and  $(h, k, m - i; i) \in \mathcal{P}$ ;
- (5)  $i < k$  and  $(h, k - i, m; i) \in \mathcal{P}$ ;
- (6)  $i < h$  and  $(h - i, k, m; i) \in \mathcal{P}$ .

**Theorem 4.2.** Let  $hkm \neq 0$ . Then  $(h, k, m; 0) \in \mathcal{N}$  iff there exists an  $i > 0$  such that one of following conditions holds:

- (1)  $i < m$  and  $(h, k, m - i, i) \in \mathcal{P}$ ;
- (2)  $i < k$  and  $(h, k - i, m, i) \in \mathcal{P}$ ;
- (3)  $i < h$  and  $(h - i, k, m, i) \in \mathcal{P}$ .

The proofs of the two theorem are similar to that of Theorem 3.4 and we omit it. The algorithm and source code for determining  $(h, k, m; x)$  to be  $\mathcal{P}$  position can be established by using the method similar to the last section. In the following algorithm the function  $\text{IsP-Pos3}(h,k,m,x)$  returns 1 if  $(h, k, m; x) \in \mathcal{P}$  and returns 0 otherwise. It is computed recursively.

**Algorithm PPos3:**

- |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ol style="list-style-type: none"> <li>(1) <math>m=1</math>.</li> <li>(2) If <math>m &gt; N</math> go to (20).</li> <li>(3) <math>k=m</math>.</li> <li>(4) If <math>k &gt; N</math> go to (19).</li> <li>(5) <math>h=k</math>.</li> <li>(6) If <math>h &gt; N</math> go to (18).</li> <li>(7) <math>x=1</math>.</li> <li>(8) If <math>2x &gt; h+1</math> go to (17).</li> <li>(9) <math>bd = \min(h, 2x)</math>.</li> <li>(10) <math>i=1</math>.</li> <li>(11) If <math>i &gt; bd</math> go to (15).</li> </ol> | <ol style="list-style-type: none"> <li>(12) If <math>i \leq m</math> and <math>\text{IsPPos3}(h, k, m-i, i)=1</math> or<br/><math>i \leq k</math> and <math>\text{IsPPos3}(h, k-i, m, i)=1</math> or<br/><math>\text{IsPPos3}(h-i, k, m, i)=1</math> go to (14).</li> <li>(13) <math>i=i+1</math>, go to (11).</li> <li>(14) <math>\text{IsPPos3}(h, k, m, x)=0</math>, go to (17).</li> <li>(15) <math>\text{IsPPos3}(h, k, m, x)=1</math>.</li> <li>(16) <math>x=x+1</math>, go to (8).</li> <li>(17) <math>h=h+1</math>, go to (6).</li> <li>(18) <math>k=k+1</math>, go to (4).</li> <li>(19) <math>m=m+1</math>, go to (2).</li> <li>(20) End.</li> </ol> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Note. The above algorithm is defined for  $x > 0$ . For  $x=0$  the steps (7), (8) and (16) should be deleted, the step (9) should be changed to  $bd=h-1$ , and in the step (12) the symbol " $\leq$ " should be replaced by " $<$ ".

To realize the algorithm PPos3, we need store all  $(h, k, m; x)$ 's amount of which is  $N \cdot N \cdot N \cdot N = N^4$ . That is, we need store  $4N^4$  integers. It takes too more memories. To save the memories we modify the algorithm in the following ways:

(i) For any position  $(h, k, m; x)$ ,  $hkmx > 0$ , we store it in a dynamic array  $\text{PPos3Arr}[m]$ , by the order  $h \geq k \geq m$ , if and only if it is a  $\mathcal{P}$  position. So, sometimes, before we manipulate a position we should sort  $h, k, m$  by descending.

(ii) For fixed  $h, k, m$ , we only store  $(h, k, m; x^*)$ , where  $x^* = \max\{x \mid (h, k, m; x) \in \mathcal{P}\}$ .

Thus, by Corollary 2.5 and Theorem 2.2 we need only store at most  $3 \sum_{m=1}^N ((N+1-m)/2)^2 = N(N+1)(2N+1)/8$  integers. Under the above conditions, the algorithm PPos3 can be realized by Theorem 4.1. In the following part of source code  $\text{InPPos3}(h, k, m, x)=1$  if  $(h, k, m; x)$ ,  $hkmx \neq 0$ , is a  $\mathcal{P}$  position, or 0 otherwise. The meaning of  $\text{InPPos2}(k, m, x)$  is the same as in Section 3.

```

typedef struct node2{int a,b,prev; } Node2;
Node2 *PPos3Arr[N];
int u[N]; /* To count the elements of each dynamic array. */
for(m=0; m<N;m++) {u[m]=0;
/* Dynamic arrays. */
  PPos3Arr[m]=(Node2 *)malloc ((N-m+1)/2*(N-m+1)/2*sizeof(Node2));
}
m0=k0=h0=0; /* The terminal position (0,0,0; x) is a  $\mathcal{P}$  position */
for(m=1; m<N; m++)
  for(k=m; k<N; k++)
    for(h=k; h<N; h++)
      for(x=1; 2*x<=h+1; x++) /* To use Theorem 2.7. */
        {flag=1;
          for(i=1; i<=(h<=2*x?h:2*x); i++)
            {k1=k-i>=m?k-i:m; m1=k-i>=m?m:k-i; /* To sort first */
              h1=h-i; h2=h1>=k?h1:k;
              k2=h2==h1?k:(h1>=m?h1:m); m2=h1>m?m:h1;
              if(i==m&&(h==k||inPPos2(h,k,i))||i==k&&
                (h==m||inPPos2(h,m,i))||i==h&&(k==m||inPPos2(k,m,i))||
                i<m&&inPPos3(h,k,m-i,i)||i<k&&inPPos3(h,k1,m1,i)||
                i<h&&inPPos3(h2,k2,m2,i))
                {flag=0;break; } /* By Theorem 4.1  $(h, k, m; x) \in \mathcal{N}$ . */
            }
          if(flag)
            {if(m==m0&&k==k0&&h==h0) /* (h, k, m; x-1) has been in PPos3Arr[m]. */
              {j=u[m]-1;
                PPos3Arr[m][j].prev=x;
                if(2*x>=h) {PPos3Arr[m][j].prev=N-1; break; } /* By Theorem 2.7. */
              }
            else /* (h, k, m; x-1) has not been in PPos3Arr[m]. */
              {j=u[m]; r[m][j].a=h; PPos3Arr[m][j].b=k; PPos3Arr[m][j].prev=1; u[m]++;
                m0=m;k0=k;h0=h; /* New  $\mathcal{P}$  position (h, k, m; 1). */
              }
            }
          else break;
        }
}

int inPPos3(int h,int k,int m,int x)
{int i;
  for(i=0; i<u[m]; i++)
    if(h==PPos3Arr[m][i].a&&k==PPos3Arr[m][i].b&&PPosArr[m][i].prev>=x) return 1;
  return 0;
}

```

Now we discuss the algorithm of winning strategy. Suppose that  $h, k, m > 1$  and  $(h, k, m; 0) \in \mathcal{N}$ . Then the Left has a winning strategy. In the algorithm there are four modules applied to search a follower of  $\mathcal{P}$  position for different  $\mathcal{N}$  position and to determine the winning move of the Left. The parameter ord is the serial number of a pile (1, 2 or 3) to be removed chips.

These modules are:

search0(ord). For an initial  $\mathcal{N}$  position  $(h, k, m; 0)$ .

search(ord). For an  $\mathcal{N}$  position  $(h, k, m; x)$  with  $x > 0$  and  $hkm \neq 0$ .

search\_1(ord). For an  $\mathcal{N}$  position  $(h, k, m; x)$  with  $x > 0$  and just one of  $h, k, m$  being zero.

search\_2(ord). For an  $\mathcal{N}$  position  $(h, k, m; x)$  with  $x > 0$  and just two of  $h, k, m$  being zero.

Besides, we define a module sort to sort its parameters by descending. In the following algorithm the function IsPPos3, IsPPos2 and IsPPos1 can be dealt with as before. And the source code is omitted.

**Algorithm mFib3Win:**

- |                                                                    |                              |
|--------------------------------------------------------------------|------------------------------|
| (1) found=0, total=h+k+m.                                          | (9) Go to (5).               |
| (2) search0(1).                                                    | (10) search(1).              |
| (3) If found=0 search0(2).                                         | (11) If found=0 search(2).   |
| (4) If found=0 search0(3).                                         | (12) If found=0 search(3).   |
| (5) If total=0 go to (23).                                         | (13) Go to (5).              |
| (6) Input R, num. /* Must be legal. */                             | (14) search_1(1).            |
| (7) If num=1 let h=h-R, if num=2 let k=k-R,<br>if num=3 let m=m-R, | (15) If found=0 search_1(2). |
| (8) If hkm≠0 go to (10),                                           | (16) Go to (5).              |
| else if hk≠0 and m=0 go to (14),                                   | (17) search_1(1).            |
| else if hm≠0 and k=0 go to (17),                                   | (18) If found=0 search_1(3). |
| else if km≠0 and h=0 go to (20),                                   | (19) Go to (5).              |
| else if h≠0 and k=m=0 search_2(1),                                 | (20) search_1(2).            |
| else if k≠0 and h=m=0 search_2(2),                                 | (21) If found=0 search_1(3). |
| else if m≠0 and h=k=0 search_2(3).                                 | (22) Go to (5).              |
|                                                                    | (23) End.                    |

**Module search0(ord):**

- |                                                                          |                                                                     |
|--------------------------------------------------------------------------|---------------------------------------------------------------------|
| (1) If ord=1 let t=h, if ord=2 let t=k,<br>if ord=3 let t=m.             | (6) sort(h0,k0,m0).                                                 |
| (2) i=1.                                                                 | (7) If IsPPos3(h0, k0, m0, i)=1 go to (9).                          |
| (3) If i>=t go to (11).                                                  | (8) i=i+1, go to (3).                                               |
| (4) h0=h, k0=k, m0=m.                                                    | (9) Let L=i, num=ord, total=total-i and found=1.                    |
| (5) If ord=1 let h0=h0-i, if ord=2 let k0=k0-i,<br>if ord=3 let m0=m0-i. | (10) If ord=1 let h=h-i, if ord=2 let k=k-i,<br>if ord=3 let m=m-i. |
|                                                                          | (11) End.                                                           |

**Module search(ord):**

- (1) If ord=1 let t=min(h, 2R), if ord=2 let t=(k, 2R), if ord=3 let t=(m, 2R).  
(2)-(11) are the same as search0(ord).

**Module search\_1(ord):**

It is the same as search(ord) except

(7) If  $\text{IsPPos2}(h_0, k_0, i)=1$  go to (9).

**Module search\_2(ord):**

(1) If  $\text{ord}=1$  let  $t=\min(h, 2R)$ , if  $\text{ord}=2$  let  $t=(k, 2R)$ , (5)  $i=i+1$ , go to (3).

if  $\text{ord}=3$  let  $t=(m, 2R)$ .

(6) Let  $L=i$ ,  $\text{num}=\text{ord}$ ,  $\text{total}=\text{total}-i$ ,  $\text{found}=1$ .

(2)  $i=1$ .

(7) If  $\text{ord}=1$  let  $h=h-i$ , if  $\text{ord}=2$  let  $k=k-i$ ,

(3) If  $i \geq t$  go to (8).

if  $\text{ord}=3$  let  $m=m-i$ .

(4) If  $i=t$  or  $\text{IsPPos1}(t-i, i)=1$  go to (6).

(8) End.

## 5. Concluding remark

It is easy to observe that the time complexity of any of the above algorithms is polynomial. The key for implementing the game is the space complexity since if  $N$  increases the memories occupied will increase by  $N$  times. How to reduce the space complexity for these algorithms further is an interesting problem.

## REFERENCES

- [1] Elwyn Berlekamp, John H. Conway, and Richard Guy, *Winning Ways*, Academic Press, Vol. 1, A. K. Peters, 2001.
- [2] J. Flanigan, "One-pile time and size dependent take-away games", *Fibonacci Quarterly* **20.1** (1982): 51-59.
- [3] J. Flanigan, "Generalized two-pile Fibonacci nim", *Fibonacci Quarterly* **16.4** (1978): 459-469.
- [4] Holshouser and Harold Reiter, "Two Pile Move-Size Dynamic Nim", *Discrete Mathematics and Theoretical Computer Science* **7** (2005): 1C10.
- [5] Holshouser and Harold Reiter, "One Pile Nim with Arbitrary Move Function", *Electronic Journal of Combinatorics* **10**(2003).
- [6] Holshouser, A. James Rudzinski and Harold Reiter, "Dynamic One-Pile Nim", *Fibonacci Quarterly* **41.3**(2003): 253-262.
- [7] Michael Zieve, "Take-away games", in *Games of No Chance*, Nowakowski, ed., Cambridge University Press, (1996): 351-361.
- [8] Peter Frankl, "The game of n-times Nim", *Discrete Mathematics*, Vol. 260, **1-3** (2003):205-09.
- [9] M. J. Whinihan, "Fibonacci Nim", *Fibonacci Quarterly*, **1.1** (1963): 9-12.