A survey on keeler's theorem and application of symmetric group for swapping game

2017 J. Phys.: Conf. Ser. 795 012048

(http://iopscience.iop.org/1742-6596/795/1/012048)

A survey on keeler's theorem and application of symmetric group for swapping game

Yohanssen Pratama[1,*] and Yohenry Prakasa[2]

[1]Faculty of Informatics and Electrical Engineering, Del Institute of Technology, Jl. Sisingamangaraja Sitoluama Laguboti Toba Samosir 22381, INDONESIA

[2]Faculty of Mathematics and Natural Sciences, Bandung Institute of Technology

* yohanssen.pratama@del.ac.id

Abstract. An episode of *Futurama* features two-body mind-switching machine which will not work more than once on the same pair of bodies. The problem is *" Can the switching be undone so as to restore all minds to their original bodies?"* Ken Keeler found an algorithm that undoes any mind-scrambling permutation, and Lihua Huang found the refinement of it. We look on the process how the puzzle can be modeled in terms group theory and using symmetric group to solve it and find the most efficient way of it. After that we will try to build the algorithm to implement it into the computer program and see the effect of the transposition notion into the algorithm complexity. The number of steps that given by the algorithm will be different and one of algorithms will have the advantage in terms of efficiency. We compare Ken Keeler and Lihua Huang algorithms to see is there any difference if we run it in the computer program, although the complexity could be remain the same.

1. Introduction

"The Prisoner of Benda", is the episode from the animated television series "*Futurama"* which explain the mechanism of two-body mind-switching machine. Any pair of person (two people) can enter the machine to swap minds, but there is some limitation that should be obeyed. The limitation is the machine will not do the swapping on the same pair of bodies. After all minds has been swapped then it appears a problem that if we want to return al off the swapping minds into its original bodies, we must search the optimum ways to return it. The television show provides an answer using a "Keeler's theorem" [1]. We will use the solution that provides by "Keeler's theorem" and an enhancement of "Keeler's theorem" that propose by Lihua Huang [1] to solve the similar case which is we usually found in the swapping game and try to build some algorithms. The algorithms will be analyzed to see its complexity and efficiency.

The problem to return all the swapping minds into the original body can be modeled using group theory. If there n bodies that involved in the experiment, then it could be write as *{1,2,...,n }*. The symmetric group S_n will consist of the *n!* permutations of *{1,2,....n }*. Because the machine switching 2 people mind each time, so we could call a permutation. A permutation which switch the mind's a into b and vice versa is called transposition, it represents 2-cycle *(ab)*. The *m*-cycle $(a_1 \ldots a_m)$ is the permutation which switches a_1's mind to a_2, a_2's mind to a_3 , ..., and a_m's mind to a_1 [1]. To divide the products in S_n into transposition (2-cycle), we compute it from right side into left side. For example, *(345) = (34)(45) = (35)(34) = (45)(35)*

The machine has limitation that it won't work more than once on the same pair of bodies, it means each transposition in S_n must be distinct. Let a product P of distinct transposition in S_n, it's represent the successive swapping of minds. Because P is a product, we can also view P as permutation, assumed that this permutation is non-trivial, otherwise it is clear that no one enter the machine. Example, suppose that 4 switches minds with 5, and then 4 switches mind with 3, this mean the product $P = (34)(45)$, or we can say in general, the mind-swapping permutation is $P = (345)$.

We must find a product α of distinct transposition such that permutation $\alpha P = I$, *because it was need* to restore all minds to their original bodies. Such transposition factors in the product α must be distinct with the product of P, because the limitation of the machine (It said that α undo P).

To find a product α that is undo the mind-scrambling permutation $P \in S_n$, we need to consider about which sequence of transposition in S_n that effected P, because they need to be distinct. Keeler's theorem provide such a product $\alpha \in S_{n+2}$. Each factor in Keeler's α contains at least one entry in the set $\{a,b\}$, where

$$a := n+1 \quad and \quad b := n+2 ;$$

It is clear that transposition factors in σ are distinct from whatever transpositions that effected P. We can say that a and b as an *outsiders* who never enter the machine during the switching process. a and b is a persons who willing as volunteers to bear another persons mind. They need in order to help another persons to restore their minds into their each original body.

P can be viewed as permutation that was expressed uniquely as the product $P = E_1 \ldots E_r$ of non-trivial disjoint cycles $E_1 \ldots E_r$ in S_n. For each $i = 1, \ldots, r$, let m_i denote the length of cycle E_i. For discussing Keeler's theorem, we assume that [1]

$$k_1 + k_2 + \ldots + k_r = n$$

2. Keeler's Method

We now describe Keeler's method for constructing a product $\sigma \in S_{n+2}$ which undoes $P = E_1 \ldots E_r$. For easier to read the notation, we write $m = m_1$, so that E_1 is a m-cycle $(u_1 \ldots u_m)$ with each $u_i \in \{1, 2, \ldots, n\}$. We can directly check that $\alpha_1 E_1 = (ab)$, where α_1 is the product of $m+2$ transposisitons given by [1]

$$\alpha_1 = (au_1)(au_2)\ldots(au_{m-1})\cdot(bu_m)(bu_m)(bu_1). \tag{1}$$

For each E_i define the similar products α_i of m_{i+2} transpositions, which satisfying

$$\alpha_i E_i = (ab) \quad \text{for} \quad i = 1, \ldots, r .$$

Notice that each transposisiton in α_i has the form (az) or (bz) for some entry $z \in E_i$.

Example 1. for $E_1 = (253)$, then by (1) we have $\alpha_1 = (2a)(5a)(3b)(3a)(2b)$, which is

$$\alpha_1 E_1 = (2a)(5a)(3b)(3a)(2b)\cdot(253) = (2)(5)(3)(ab) = (ab)$$

We know that disjoint cycles are commute, so (ab) commutes with every transposition in S_n, then

$$\varphi := \alpha_r \ldots \alpha_2 \alpha_1$$

is a product of distinct transpositions for which $\varphi P = (ab)^r$. So we can conclude as,

$$\alpha = \begin{cases} (ab)\varphi\,, & \text{if } r \text{ is odd} \\ \varphi\quad, & \text{if } r \text{ is even} \end{cases} \tag{2}$$

Clearly that α undoes P and α is a product of distinct transpositions in S_{n+2}, each containing at least one entry in a , b, as desired .

From equation (1) and (2), we can see that the number of factors α will be either $n+2r+1$ or $n+2r$. It depend on r, whether it was odd or even. However, for each $r > 2$, the number of factors that needed to undo P can be reduced. For example, for $r = 3$, $P = (12)(34)(56)$ is undone by Keeler's product of 13 transpositions

$$\alpha = (ab)(5a)(6b)(6a)(5b)(3a)(4b)(4a)(3b)(1a)(2b)(2a)(1b),$$

but in reality P can be undone by the product of 11 transpositions,

$$(5a)(6b)(6a)(5b)(1a)(2a)(3a)(4b)(4b)(3b)(1b).$$

After we see the brief of description that become a core of Keeler's algorithm itself, now we will try to build some computer algorithm from it. For the Keeler's algorithm we tried to define the pseudocode as below:

```
1     BEGIN
2       swap(source, destiny, array) {
3       if swap = 1, return -1;
4         swap source with destiny;
5       if all mind already restored to original body, return 1;
7         for (source = 0, source = array.length -1,i++)
8             for (destiny = 0, destiny = array.length-1, j++)
9           {
10            if ((source != destiny) && (swap(source, destiny) == 1);
11            {
12                print swap;
13              return 1;
14          }
15        }
16    }
17      END
```

If we see the pseudocode above, the body number will be stored in array and if the swap was already tried it will return an invalid value. If the source and destination already swapped the records will be updated so it can't be swapped again. When everyone mind already comeback to original body the search will be end. In the next section, we will look at LiHua Huang proposed theorem that is the refinement from the Keeler's theorem. The theorem showing that P can be undone only by $n+ r +2$ distinct transpositions that each containing at least one entries a, b which is better than n + 2r.

3. Analysis of optimal refinement of Keeler's method

Keeler's algorithm was designed to undo every mind-scrambling permutation $P = E_1...E_r$ that is affected by an unknown sequence of mind swaps. In this section, we will see another algorithm to return all the scramble minds into it original bodies by Lihua Huang.

Theorem 1. Let $P = E_1 \ldots E_r$ be a product of disjoint m_i-cycles $E_i \in S_n$, with $m_i \geq 2$ and $n = m_1 + \ldots + m_r$. Now we define $a = n+1$ and $b = n+2$. Then P can be undone by a product γ of $n+r+2$ distinct transpositions in S_{n+2}, each containing at least one entry in a, b.

Proof. Write m = m_1, so that E_1 is a m-cycle $(u_1 \ldots u_m)$. We could define the product that corresponding to the E_1 cycle

$$G_1(a) = (u_1\, a)(u_2\, a)\ldots(u_m\, a)$$

which the most left factor is

$$F_1(a) = (u_1\, a)$$

Corresponding to each cycle E_i for $i = 1, \ldots, r$, we could define $G_i(a)$ and $F_i(b)$ similarly and set,

$$\gamma = (ab)\cdot G_r(a)\ldots G_2(a)\cdot(u_m\, a) G_1(b)(u_1\, a)\cdot F_2(b)\ldots F_r(b)$$

It is quite easy to check that γ undoes P and that γ is a product of $n+r+2$ distinct transpositions in S_{n+2}, each containing at least one entry in $\{a,b\}$.

Example 2. Let $P = (12)(345)(67) \in S_7$. Find $\sigma \in S_9$, a product of distinct transpositions such that each transposition contains at least one of the entries a, b and that α undoes P. Keeler's method:

$$\alpha_1 = (ab)(a1)(b2)(a2)(b1)(a3)(a4)(b5)(a5)(b3)(a6)(b7)(a7)(b6).$$

There are $n+2r+1 = 7 + 6 + 1 = 14$ transpositions in α_1, which undoes P.
LiHua Huang refinement algorithm [1]:

$$\alpha_2 = (ab)(6a)(7a)(3a)(4a)(5a)(2a)(1b)(2b)(1a)(3b)(6b).$$

Notice that there are only $n+r+2 = 7+3+2 = 12 < 14$ transpositions in α_2, which undoes P.

Example 3. Let $P = (12) \in S_2$. Undo P with $\sigma \in S_4$. Keeler's method:

$$\alpha_1 = (ab)(a1)(b2)(a2)(b1).$$

There are $n + 2r + 1 = 2 + 2 + 1 = 5$ transpositions in α_1, which undoes P. LiHua Huang refinement algorithm:

$$\alpha_2 = (ab)(2a)(1b)(2a)(1b).$$

There are $n + r + 2 = 2 + 1 + 2 = 5$ transpositions in α_2, which also undoes P. (Observe that if one switches the disjoint transpositions $(a1)$ and $(b2)$ in α_1 and then renames a, b as b, a, respectively, one can see that α_1 becomes α_2).

From example above, we can conclude that when $r = 1, 2$, both Keeler's method and LiHua Huang algorithm are best possible. However, when r is large, LiHua Huang said have showed that $n + r + 2$ is much less than $n+2r$.

If we look the pseudocode again, then algorithm the complexity will be:

$$\sum_{i=0}^{t-1}\sum_{j=0}^{t-1} 1 = t.t = t^2$$

The algorithm contains two nested loops (first for and second for) which the segment inside loops are executed consecutively t-1 times (t is the total numbers of the input). The complexity of the algorithm will be O(t^2) where t is the length of array. Because both Keeler's and LiHua Huang algorithm has same n instead r and the leading constant in 2r may be ignored, the complexity still leads to O(t^2).

4. Conclusion

From our survey, we can conclude that from simple swapping game, we can search the optimal way to solve the problem. Although the complexity are same for both Keeler's and Lihua Huang algorithm that is O(t^2), but the process that given by the Lihua Huang algorithm is more efficient. For future works, we plans to find another way to utilize the advantages that owned by the Lihua Huang algorithm to improve the running time in the computer program. We hope science can help us solve our daily problem especially on programming or computation problem.

References

[1] Evans R and Huang L 2012 *Keeler's theorem and products of distinct transpositions*
[2] Isaacs M 1994 *Algebra: A Graduate Course,* Graduate Studies in Mathematics vol 100 Amer. Math. Soc., Providence, RI
[3] Bassil Y A 2012 Comparative Study on the Performance of Permutation Algorithms *Journal of Computer Science and Research* **1** 7 -19
[4] Levine A 2010 The Futurama of physics with David X. Cohen. *Amer. Physical Soc. News 19* **3**
[5] Shanker S 2016 Time Complexity of Matrix Transpose Algorithm using Identity Matrix as Reference Matrix *International Journal of Computer Science and Information Technologies* **5** 2347-48
[6] Cormen T Leiserson C Rivest R and Stein C 2009 *Introduction to Algorithms 3rd ed* (MIT Press)
[7] Levitin A 2006 *Introduction to the Design and Analysis of Algorithms 2nd ed* (Addison Wesley)
[8] Donald K 1998 *The Art of Computer Programming (Sorting and Searching)* vol 3
[9] Bona M 2004 *Combinatorics of Permutations* (Chapman Hall-CRC)
[10] Oliver R 2008 On the parity of permutation *Amer. Math. Monthly* **118** 734-35
[11] Neuenschwander D 2001 On the representation of permutations as products of transpositions *Elem. Math.* **56** 1-3
[12] Grime J 2010 *Futurama and Keeler's Theorem : original edit, video exposition of Futurama's episode "The Prisoner of Benda"* available at http://www.youtube.com/watch?v=8M4dUj7vZJc