# Irving's Algorithm Revisit: An Implementation in Python

ZHANG Shiyu

October 6, 2020

## 1 Introduction

The *stable roommate problem* is a problem involving 2n people, trying to find a stable *roommate assignment* **A** that match the 2n people into n disjoint pairs. A roommate assignment **A** is called *unstable* if there is a *blocking pair*, such that each of them prefer the other to his assigned roommate in **A**. An assignment **A** that contains no blocking pair is called *stable*. An instance of stable roommate problem is called *solvable* if it has at least one stable assignment. An efficient algorithm to detect whether a given instance of stable roommate problem is solvable and to find one stable assignment if it exists, is first proposed by Irving (1985). The following of this article would present an implementation of Irving's algorithm in Python for the pedagogical purpose of facilitating interesting readers to understand this algorithm and relevant issues about stable roommate problem.

**Claim 1.** *Not all stable roommate problem are solvable.*

*Proof.* An easy way to construct a stable roommate problem of 2n people that is unsolvable is as following: assign one person, say $x$, as the tail entry in each of others' preference list, let the first entry of the other (2n-1) people be a permutation of them. So, in any assignment **A**, the person $i$, who is paired with $x$, is the head entry of someone $j, j \neq x$. And $j$ would therefore prefer $i$ to his current partner in **A**, while $i$ would also prefer $j$ to his current partner $x$ since $x$ is the tail entry in his list. $(i, j)$ form a blocking pair and **A** is not stable. The following is an example of a unsolvable roommate problem of size 6.

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | ... | ... | ... | 6 |
| 2 | 3 | ... | ... | ... | 6 |
| 3 | 4 | ... | ... | ... | 6 |
| 4 | 5 | ... | ... | ... | 6 |
| 5 | 1 | ... | ... | ... | 6 |
| 6 | ... | ... | ... | ... | ... |

Table 1: A unsolvable Roommate Problem of 6 people

□

# 2 Part 2

To keep our discussion consistent, we will first go through some definitions that would be used later. Then I would present the algorithm, both in the form of pseudo code and implementation in Python, meanwhile, some discussions and proofs would be given to justify the correctness of the algorithm and necessity of certain line of codes.

## 2.0 The Definitions

**Table:** The current set of preference lists at any stage of the algorithm is called a table;

$\boldsymbol{e_i, s_i, h_i}$ **:** Let $e_i$ denote a person. For any given stage of the algorithm, let $h_i$ denote the current head of person $e_i$'s list and let $s_i$ denote the current second entry on $e_i$'s list.

**Semi-engaged:** A person $e_i$ is said to be semi-engaged to $h_i$ if.f. $e_i$ is the last entry on $h_i$'s list.

**Free:** A person who is not semi-engaged is called free.

**Exposed Rotation:** In a Table, an exposed rotation R is an ordered subset of people $E = \{e_1, e_2, \cdots, e_r\}$, such that $s_i = h_{i+1}$ for all $i$ from 1 to r,where $i+1$ is taken module r. Moreover, let H denote the set of head entries of E ordered according to the order of E, S denote the set of second entries with corresponding order, and write R = (E, H, S).

**Elimination of R:** For an exposed rotation R = (E, H, S) in a table T, the elimination of R from T takes the following operation: for every $s_i$ in S, remove every entry below $e_i$ from $s_i$'s list in T, then remove $s_i$ from k's list for every k who is just removed from $s_i$'s list.

**Contained in:** A roommate assignment A is said to be contained in T if.f. for each pair $(i, j)$ in A, $i$ is on $j's$ list and $j$ is on $i's$ list.

**Tail, Body:** Let $e_1$ denote the person who is visited twice in the procedure of seeking an exposed rotation, i.e. where the cycle is detected. Every person who is visited before $e_1$ is said to be on a tail of R, and the other people in the body of R.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 3 | 7 | 8 | 6 | 4 | 2 | 5 |
| 2 | 3 | 6 | 1 | 7 | 8 | 5 | 4 |
| 3 | 8 | 6 | 7 | 4 | 1 | 5 | 2 |
| 4 | 8 | 2 | 5 | 1 | 3 | 7 | 6 |
| 5 | 4 | 1 | 6 | 7 | 3 | 8 | 2 |
| 6 | 1 | 5 | 4 | 3 | 2 | 8 | 7 |
| 7 | 1 | 8 | 6 | 5 | 4 | 2 | 3 |
| 8 | 7 | 2 | 5 | 3 | 4 | 1 | 6 |

Table 2: An instance of roommate problem for 8 people

| | | | |
|---|---|---|---|
| 1 | 3 | 7 | |
| 2 | 6 | 8 | 4 |
| 3 | 8 | 6 | 1 |
| 4 | 2 | 5 | |
| 5 | 4 | 6 | |
| 6 | 5 | 3 | 2 |
| 7 | 1 | 8 | |
| 8 | 7 | 2 | 3 |

Table 3: Preference lists after phase-I execution

(1, 7), (2, 8), (3, 6), (4, 5)
(1, 7), (2, 6), (3, 8), (4, 5)
(1, 3), (2, 4), (5, 6), (7, 8)

Table 4: Three distinct stable assignments contained in the table

## 2.1 The Algorithm

Irving's algorithm can be divided into two phases:

### 2.1.1 Phase-I

**Algorithm** Phase-I of Irving's algorithm

**Input:** A table of preference lists of the 2n people.

**Output:** A table where every person is semi-engaged or none.

**while** Not every people has his proposal held by someone **do**<br>
    **if** There is any person whose preference list is empty. **then**<br>
        **return** None $\leftarrow$ Terminate since no stable roommate assignment exists.<br>
    **end if**<br>
    Pick an arbitrary person $x$ whose proposal has not been held by someone.<br>
    Let him *propose* to the head entry $y$ in his current preference list.<br>
    **if** $y$ prefer $x$ to the tail entry in his preference current list **then**<br>
        Let $y$ *hold* $x$'s proposal and remove anyone ranked below x from his list.<br>
    **else**<br>
        Let $x$ remove $y$ from his current preference list.<br>
    **end if**<br>
**end while**<br>
**return** A table where every person is semi-engaged. $\leftarrow$ Move into the phase-II of Irving's algorithm.

### 2.1.2 Correctness of Phase-I

Some lemmas from Irving's paper to be added here soon.

### 2.1.3 Phase-II

**Algorithm** Phase-II of Irving's algorithm

**Input:** A table of preference lists of the 2n people, where everyone is semi-engaged.

**Output:** A stable roommate assignment or none.

1: **while** Someone whose preference list has more than one entry and no one has empty list **do**
2:     Find an exposed rotation and eliminate it.
3:     **if** There is any person whose preference list is empty. **then**
4:         **return** None ← Terminate since no stable roommate assignment exists.
5:     **end if**
6: **end while**
7: **return** A table where every person has only one entry in his preference list. ¡- By pairing each person with the unique entry in his list, we get a stable roommate assignment.

### 2.1.4 Correctness of Phase-II

A question that might come into our mind when we read step 2 of the above pseudo-code is **does there always exist a rotation exposed?** And if it does exist, **how shall we find it?** The following lemma suggest the existence of an exposed rotation whenever there is someone whose list has two or more entries, and provides a method to find it.

**Lemma 2.1.** [G] *(In phase 2) If $T$ is a table where no list is empty, and at least one person has more than one entry, then there is a rotation exposed in $T$.*

*Proof.* Assume there are more than one entry on $e_i$'s list and denote the head and second entry as $h_i$, $s_i$ respectively, we know that $e_i$ must be the bottom entry of $h_i$'s list and also be on $s_i$'s list.

We know that $e_i$ could not be the bottom entry on $s_i$'s list, as after phase 1, one could not be the bottom entry of more than one people's lists at the same time $\Rightarrow$ there are at least two entries on $s_i$'s list and we assume the bottom entry is $e_j$, $j \neq i$ (i.e. $s_i$ is the head entry of $e_j$). Since $e_j$ not being $s_i$'s head entry $\Rightarrow$ $s_j$ would be the bottom entry of someone other than $e_j$, and $s_i$ would not be both the head and the bottom entry in $e_j$'s list $\Rightarrow$ there are at least two entries in $e_j$'s list, and we denote the second entry as $s_j$.

By exact argument as above, we know there are more than one entry in $s_j$'s list, and the bottom entry of it must be different from $e_j$, we can name it $e_k$ and argue that there are more than one entry on $e_k$'s list $\cdots$ Since there are only finite people, the repetition of above argument must lead to a cycle, and we find an exposed rotation. □

$$\begin{array}{c|cccc}
e_i & h_i & s_i & \ldots & \\
\ldots & \ldots & \ldots & & \\
s_i & \ldots & e_i & \ldots & e_j \\
\ldots & \ldots & \ldots & & \\
e_j & s_i & s_j & \ldots & \\
\ldots & \ldots & \ldots & & \\
s_j & \ldots & e_j & \ldots & e_k
\end{array}$$

$\ldots$ means there might or might not exist an entry

**Corollary 2.1.** [$G$] *If e is a person with two or more entries on its list in table T, then e is either in a tail or in the body of a rotation exposed in T.*

The above lemma indicates that we could start from any person whose list contains more than one entry in T and the finiteness of the table would guarantee that we could find an exposed rotation.

As we purposely reduce the table by repeatedly eliminating exposed rotations, a cautious reader might worry **would the rotation elimination delete some stable assignments contained, or even all of them, such that no stable assignment contained in the remaining table?** Actually, we might lose some, but not all of the stable assignments, and such a worry is assured by the following two lemmas.

**Lemma 2.2.** [$G$]*Let R be a rotation exposed in table T, A be a stable assignment contained in T. If $e_i \in E$ and $e_i$ is paired with $h_i$ in A, then $(e, h)$ must also be a pair in assignment A, for any $(e, h)$ in either the body or a tail of R.*

*Proof.* Assume $e_i$ is a person in a rotation R exposed in T and he is paired with $h_i$ (i.e his head entry) in a stable assignment A contained in T. Then we consider $e_{i-1}$

(module r) in R, we have $s_{i-1} = h_i$ (i.e. the second entry of $e_{i-1}$ is also the head entry of $e_i$). If $e_{i-1}$ is not paired with $h_{i-1}$ in A, neither would $e_{i-1}$ be paired with $s_{i-1}$ since $s_{i-1}$ is paired with $e_i$ in A $\Rightarrow$ $e_{i-1}$ would prefer $s_{i-1}$ to his current partner. Moreover, $s_{i-1}$ is on $e_{i-1}$'s list, $\Rightarrow$ $e_{i-1}$ would also appear on $s_{i-1}$'s list, while $e_i$ being the bottom entry in $s_{i-1}$'s list $\Rightarrow$ $s_{i-1}$ would prefer $e_{i-1}$ to $e_i$ $\Rightarrow$ $(e_{i-1}, s_{i-1})$ forming a blocking pair in stable assignment A, a contradiction! $\Rightarrow$ $e_{i-1}$ must be paired with $h_{i-1}$ in A.

Since the above argument goes backwardly and the rotation is cyclic, we know for any $e_j$ in the body of the rotation, he would be paired with his head entry $h_j$ in assignment A. Moreover, given a tail of R, let us denote the last person in the tail as $p$ and his second entry in list as $q$, we know $q = h_1$ (i.e. the second entry in $p$'s list is also the head entry in $e_1$'s list, where $e_1$ is the first person in the rotation), so we can again apply the above argument backwardly for each person in the tail, and claim that every person in the tail is also paired with their head entry in list in assignment A. $\square$

$$
\begin{array}{c|ccccc}
e_i & h_i & s_i & \dots & & \\
\dots & \dots & \dots & & & \\
h_i & \dots & e_{i-1} & \dots & e_i & \\
\dots & \dots & \dots & & & \\
e_{i-1} & h_{i-1} & h_i & \dots & &
\end{array}
$$

The contradiction is given by:

$$s_{i-1} \succ_{e_{i-1}} \varphi_A(e_{i-1}), \quad e_{i-1} \succ_{s_{i-1}} e_i$$

$\varphi_A(e_{i-1})$ is $e_{i-1}$'s partner and $\varphi_A(e_{i-1}) \neq h_{i-1}$,

**Lemma 2.3.** [G]*If rotation $R = (E, H, S)$ is exposed in T, and there exists a stable assignment in T where $e_i \in E$ is paired with $h_i$, then there also exists a stable assignment in T where $e_i$ is not paired with $h_i$ and by Lemma 2.2, no $e_j$ is paired with $h_j$ for every $e_j \in E$.*

*Proof.* 1) If the interaction between set of people and set of head entries in R is not empty, i.e. $E \cap H \neq \emptyset$, for a person $e_1$ in $E \cap H$, where $e_i = h_j$ for some people $e_j$ in the body of the rotation, since there are more than one entries in $e_i$'s list, (otherwise he would not be in the rotation), $\Rightarrow$ $h_i \neq e_j$ $\Rightarrow$ $e_i$ could not be paired with $e_j$ and $h_i$ at the same time in an assignment A, and by Lemma 2.2, no person in the rotation is paired with his head entry.

2)If $E \cap H = \emptyset$, assume that there exists a stable assignment A, where $e_i$ is paired

with $h_i$ in A, for another assignment A', where $e_j$ is paired with $s_j$ for each $e_j$ in the rotation, and anyone not in $E \cup H$ is paired with his partner as in A, we claim that A' is also stable.
Since $h_j$ would prefer his partner $e_{j-1}$ in A' to $e_j$ in A, if there is any blocking pair, it must be involved with some $e_k$ in E (i.e. no instability come from people in H). Assume that there is a blocking pair $(e_k, x)$ to A', where $e_k$ prefers $x$ to $s_k$, and $x$ also prefers $e_k$ to his current partner in A', since $x \neq h_k$, we know $x$ is not in $e_k$'s list in T.

- i)If $x$ rejects $e_k$ in phase-I and got himself dropped from $e_k$'s list, (note that it must be $x$ to reject $e_k$ instead of $e_k$ rejecting $x$, otherwise $e_k$ should have already dropped anyone ranked below $x$, including $h_k$ and $s_k$ ) he must hold the proposal from someone whom is considered better than $e_k$ by him, and he would prefer anyone on his current list in T to $e_k$, so in A' he must be paired with someone better than $e_k$ and would not form a blocking pair with $e_k$;

- ii)If $x$ removes $e_k$ from his list in phase-II, there are two cases of reduction in phase-II,

  - a) one is that $x$ be in $H$ for a rotation R, and $e_k$ was his head entry at that time, he then rejected $e_k$ and proposed to his second entry. In this case, $x$ was already the bottom entry in $e_k$'s list, $s_k$ had been dropped from $e_k$'s list before $s_k$ was dropped, a contradiction to our assumption!

  - b) the other is that $x$ be in $S$ for a rotation R, i.e. $x$ was the second entry of some $e_l$ in E, and $e_l$ rejected $h_l$ and proposed to $x$. In this case, $e_k$ lies behind $e_l$ in $x$'s list and get dropped. In assignment A', $x$ would be paired with someone no worse than $e_k$, so he would prefer his current in A' to $e_k$, no blocking pair would be form.

By above argument, we know no one would form blocking pair with $e_k$ for any $e_k$ in E. Thus, A' is also stable.

Case a

$$\begin{array}{c|ccccc}
x & e_k & \cdots & \cdots & \cdots & \\
\vdots & \cdots & & & & \\
e_k & \cdots & \cdots & x & \cancel{\cdots} & \cancel{s_k}
\end{array}$$

Case b

$$\begin{array}{c|ccccc}
e_l & \cdots & x & \cdots & \cdots & \\
\vdots & \cdots & & & & \\
x & \cdots & e_l & \cdots & e_k & \cdots \\
\vdots & \cdots & & & & \\
e_k & \cdots & x & \cdots & h_k & s_l
\end{array}$$

□

Lemma 2.3 assure us that when an elimination of rotation delete a contained stable assignment A, we can always construct another stable assignment A' by pairing each one in the rotation with his second entry and pair anyone not involved in the rotation with his original partner in A. Lastly, we might wonder, when the algorithm terminates with a solution, **is the solution generated by the algorithm always a stable assignment?** Lemma 2.4 answers this question.

**Lemma 2.4.** [G]*If the algorithm ends with a single entry on each list, then pairing each person to that entry gives a stable assignment.*

*Proof.* Suppose $(x, y)$ form a blocking pair to the assignment represented by the single-entry table, where $y$ prefer $x$ to the unique entry in his list while $x$ is not in $y$'s list. However, as we have proven in part 2) of the proof for lemma 2.3, no such blocking pair exist, so the assignment represented by the single-entry table is a stable one. □

## 2.2 Can we use Irving's algorithm to find all the stable assignments contained?

**Theorem 2.1.** *If A is any stable roommate assignment, then there is an execution of Irving's algorithm that produces A.*

*Proof.* Firstly, we know that, for any given initialization of the preference lists, the resulting table after phase-I is determined, i.e. it is invariant with choice of the first person to propose. This is implied by Lemma 1 in Irving's paper:

> LEMMA 1. If y rejects x in the proposal sequence described above (phase-I of Irving's algorithm), then x and y cannot be partners in a stable matching.

which says that the remaining entries in $p$'s list after phase-I are possible partners of $p$ in some stable matching, since this possibility would not be affected by the choice of first person to propose, we know the resulting table after phase-I is determined. Then, we want to show that, for any table T obtained from the partial execution of Irving's algorithm **in phase-II**, if a stable assignment A is contained in T, then either A is represented by T, (where there is a unique entry on each person's list) or A would remain contained in the table after further execution.

- If, for each person $p_i$, the head entry in $p_i$'s list is his partner in A $q_i$, we know for each pair $(p_j, q_j)$ in A, $p_j$ is the head entry in $q_j$'s list $\Rightarrow$ $q_j$ is both the bottom entry and head entry in $p_j$'s list $\Rightarrow$ there is only one entry in $p_j$'s list. We apply the above argument for each person and deduce that A is the stable assignment represented by T;

- If there is someone $p_i$, whose partner in A $q_j$ is not the head entry in his list $\Rightarrow$ there are at least two entries in $p_i$'s list $\Rightarrow$ by Corollary 2.1, $p_i$ is either in the body or tail of a rotation exposed in T. $\Rightarrow$ by Lemma 2.5, we know any person $e_i$ in the body of that rotation would not be partnered with the head entry $h_i$ in A, otherwise, $p_j$ would also be paired with his head entry in A, contradicts with our assumption.

  When the rotation R is eliminated,

  - Each $e_i$ in the body of R is forced to drop his head entry $h_i$ and proposed to the current second entry $s_i$, since no $e_i$ is paired with $h_i$ in A, it is safe to drop them;

  - Meanwhile, every $s_i$ is forced to reject $e_{i+1}$'s proposal and accept $e_i$'s proposal, dropping anyone below $e_i$ in his list. Since the above paragraph implies that $e_i$ could only be paired with someone no better than $s_i$ in A $\Rightarrow$ if $s_i$ is paired with someone below $e_i$ in his list, then $s_i$ would prefer $e_i$ to his current partner and $e_i$ would also prefer $s_i$ to his current partner, forming a blocking pair $\Rightarrow$ $s_i$ would not be paired with anyone below $e_i$ in his list in A $\Rightarrow$ It is safe to drop anyone below $e_i$ in his list.

  By above, we know A remain contained in the table after elimination of R.

$\square$

**Corollary 2.2.** *Let $R$ be an exposed rotation in table $T$, and let $T'$ be the table after eliminating $R$ from $T$, then $T'$ contains all stable assignments that $T$ contains, except for those assignment where $e_i$ is paired with $h_i$ and $(e_i, h_i)$ is in $R$.*

## 2.3 The Code

Phase-I of Irving's algorithm

```
import numpy as np
import random
import matplotlib.pyplot as plt

ENABLE_PRINT = 0
DETAILED_ENABLE_PRINT=0
#convert the preference matrix into ranking matrix
def get_ranking(preference):
    ranking = np.zeros(preference.shape,dtype=int)
    for row in range(0,len(preference[:,0])):
        for col in range(0,len(preference[0,:])):
            ranking[row,col]=list(preference[row,:]).index(col)
    return ranking


def phaseI_reduction(preference, leftmost, rightmost, ranking):
    ## leftmost and rightmost is updated here
    set_proposed_to=set() ## this set contains the players who has been proposed to and holds someone
    for person in range(0,len((preference[0,:]))):
        proposer = person
        while True:
            next_choice = preference[proposer,leftmost[proposer]]
            current = preference[next_choice,rightmost[next_choice]]

            while ranking[next_choice,proposer]> ranking[next_choice,current]:
                ## proposer proposed to his next choice but being rejected
                if ENABLE_PRINT and DETAILED_ENABLE_PRINT: print("player", proposer+1, "proposed to", next_choice+1, "; ", next_choice+1, "rejects", proposer+1 )
```

```python
                leftmost[proposer] = leftmost[proposer] + 1 ##proposer's preference list got reduced by 1 from the left
                next_choice = preference[proposer, leftmost[proposer]]
                current = preference[next_choice, rightmost[next_choice]]

            ## proposer being accepted by his next choice and next choice rejected his current partner
            if current!= next_choice: ##if next choice currently holds somebody
                if ENABLE_PRINT and DETAILED_ENABLE_PRINT: print("player", proposer + 1, "proposed to", next_choice + 1,"; ", next_choice + 1, "rejects", current + 1, " and holds", proposer+1 )
                leftmost[current]=leftmost[current]+1
            else: ##if next choice currently holds no body
                if ENABLE_PRINT and DETAILED_ENABLE_PRINT: print("player", proposer + 1, "proposed to", next_choice+1, "; ", next_choice+1, "holds", proposer+1)

            rightmost[next_choice] = ranking[next_choice, proposer] ##next choice's preference's list got reduced, rightmost is proposer now

            if not (next_choice in set_proposed_to): ##if no one is rejected <=> next choice has not been proposed before proposer proposed
                break
            proposer = current ##the one who being rejected is the next proposer
        set_proposed_to.add(next_choice)

    soln_possible = not (proposer==next_choice)

    if ENABLE_PRINT: print("The table after phase-I execution is:")
    if ENABLE_PRINT: friendly_print_current_table(preference, leftmost, rightmost)
    return soln_possible, leftmost, rightmost

def get_all_unmatched(leftmost, rightmost):
    unmatched_players = []
```

```
    for person in range(0, len(leftmost)):
        if leftmost[person] != rightmost[person]:
            if ENABLE_PRINT and DETAILED_ENABLE_PRINT: print(person + 1, "is unmatched")
            unmatched_players.append(person)
    return unmatched_players


def update_second2(person,preference, second, leftmost, rightmost, ranking):
    second[person]=leftmost[person]+1 #before updation, second is simply leftmost +1
    pos_in_list = second[person]
    while True: # a sophisticated way to update the second choice, as some person between leftmost and rightmost might be dropped as well
        next_choice = preference[person, pos_in_list]
        pos_in_list += 1
        if ranking[next_choice, person] <= rightmost[next_choice]: # check whether person is still in next_choice's reduced list <=> next_choice is still in his list
            second[person] = pos_in_list -1
            return next_choice, second

def seek_cycle2(preference, second, first_unmatched, leftmost, rightmost, ranking):
    #tail= set()
    #print("I am in seek_cycle2")
    cycle =[]
    posn_in_cycle = 0
    person = first_unmatched
    if ENABLE_PRINT and DETAILED_ENABLE_PRINT: print("p_",posn_in_cycle+1,":",person+1)

    while not (person in cycle): ##loop until the first repeat
        cycle.append(person)
        posn_in_cycle+=1
        next_choice, second = update_second2(person,preference, second, leftmost, rightmost, ranking)
        if ENABLE_PRINT and DETAILED_ENABLE_PRINT: print("q_",posn_in_cycle,":",next_choice+1)
```

```
        person = preference[next_choice,rightmost[next_choice]]
        if ENABLE_PRINT and DETAILED_ENABLE_PRINT: print("p_",posn_in_cycle+1,":",person+1)
    #after this loop, person is the one who repeats first

    last_in_cycle= posn_in_cycle-1 #position of the last one in cycle in the "cycle" list
    #tail = set(cycle) #using the set object in Python, we don't need cycle_set
    while True: #this is used to find the head of the cycle and its position in the "cycle" list
        posn_in_cycle = posn_in_cycle - 1
        #tail = tail.remove(cycle[posn_in_cycle])
        if cycle[posn_in_cycle]==person: #loop until we get the person who repeat first
            first_in_cycle = posn_in_cycle
            break
    #print("!!!",first_in_cycle,last_in_cycle)
    #print("I am out of seek_cycle2 now")
    friendly_print_rotation(cycle, first_in_cycle, last_in_cycle, preference, leftmost, second)
    return first_in_cycle, last_in_cycle, cycle, second

def phaseII_reduction2(preference, first_in_cycle, last_in_cycle, second, leftmost, rightmost, soln_possible, cycle):
    #print("I am in phase ii reduction2")
    #print("input is:")
    #print([ leftmost, rightmost, second])
    for rank in range(first_in_cycle, last_in_cycle+1):
        proposer = cycle[rank]
        leftmost[proposer] = second[proposer]
        second[proposer] = leftmost[proposer]+1 #it is mentioned that proper initialization is unnecessary
        next_choice = preference[proposer,leftmost[proposer]]
        if ENABLE_PRINT and DETAILED_ENABLE_PRINT: print(proposer+1, "proposed to his second choice in the reduced list:", next_choice+1, ";", next_choice+1,"accepted ", proposer+1, "and rejected", preference[next_choice,rightmost[next_choice]]+1 )
        rightmost[next_choice] = get_ranking(preference)[next_choice,proposer]
```

```
    #print([leftmost, rightmost, second])
    #To check whether stable matching exists or not#
    rank = first_in_cycle
    while (rank <= last_in_cycle) and soln_possible:
        proposer = cycle[rank]
        soln_possible = leftmost[proposer] <= rightmost[proposer]
        rank+=1
    if not soln_possible:
        if ENABLE_PRINT: print("No stable matching exists!!!")
        return soln_possible, first_in_cycle, last_in_cycle, second.copy(), leftmost.copy(), rightmost.copy(), cycle

    #A special step to handle the case of more than one cycle, seems not contained in the code in paper#
    for person in range(first_in_cycle, last_in_cycle):
        if leftmost[cycle[first_in_cycle]] != rightmost[cycle[first_in_cycle]]:
            to_print =np.array(cycle[first_in_cycle:last_in_cycle + 1])+1
            if ENABLE_PRINT and DETAILED_ENABLE_PRINT: print("E=",to_print, " is still unmatched")
            if ENABLE_PRINT: print("The table after rotation elimination is:")
            if ENABLE_PRINT: friendly_print_current_table(preference, leftmost, rightmost)
            return soln_possible, first_in_cycle, last_in_cycle, second.copy(), leftmost.copy(), rightmost.copy(), cycle
    to_print = np.array(cycle[first_in_cycle:last_in_cycle + 1]) + 1
    if ENABLE_PRINT and DETAILED_ENABLE_PRINT: print("E=",to_print, "is all matched")
    first_in_cycle=0

    #print("I am out of phase II reduction2 now")
    if ENABLE_PRINT: print("The table after rotation elimination is:")
    if ENABLE_PRINT: friendly_print_current_table(preference, leftmost, rightmost)
    return soln_possible, first_in_cycle, last_in_cycle, second.copy(), leftmost.copy(), rightmost.copy(), cycle

def friendly_print_current_table(preference, leftmost, rightmost):
    for person in range(0,len(preference)):
```

```
        to_print = []
        for entry in range(leftmost[person],rightmost[person]+1):
            if get_ranking(preference)[preference[person, entry],person]<=
                rightmost[preference[person,entry]]:
                to_print.append(preference[person,entry])
        to_print=np.array(to_print)
        print(person+1,"|",to_print+1)

def friendly_print_rotation(cycle,first_in_cycle,last_in_cycle, preference,
    leftmost,second):
    print("The rotation exposed is:")
    print("E| H S")
    for person in range(first_in_cycle,last_in_cycle+1):
        print("{0}| {1} {2}".format(cycle[person]+1,preference[cycle[person],
            leftmost[cycle[person]]]+1,preference[cycle[person],second[cycle[
            person]]]+1))

def friendly_print_sol(partners):
    seen = []
    pairs=[]
    to_print = []
    for sol in partners:
        for people in range(0, len(sol)):
            if people not in seen:
                seen.append(people)
                pairs.append((people+1,sol[people]+1))
                seen.append(sol[people])
        to_print.append(pairs)
        pairs = []
        seen=[]
    return to_print


def Find_all_Irving_partner(preference):

    ranking = get_ranking(preference)
    leftmost = np.zeros(len(preference[0, :]), dtype=int) #leftmost indicates
         the position of the person who holds i's proposal
    second = np.zeros(len(preference[0, :]), dtype=int) + 1
```

```
    rightmost = np.zeros(len(preference[0, :]), dtype=int) + len(preference
        [0,:]) - 1 #rightmost indicates the position of the person whose
        proposal i holds
    partner = np.zeros(len(preference[0, :]), dtype=int)
    soln_possible = False
    first_unmatched = 1
    first_in_cycle = 0
    last_in_cycle = 0
    cycle=[]
    partners = []
    soln_found = False

    if ENABLE_PRINT: print("The preference lists are:")
    if ENABLE_PRINT: print(preference+1)

    soln_possible, leftmost, rightmost = phaseI_reduction(preference,
        leftmost, rightmost, ranking)
    if not soln_possible:
        if ENABLE_PRINT: print("No stable matching exists!!")
        return partners
    second = leftmost + 1



    seen = []
    queue =[]
    qlfmost =leftmost.copy()
    qrtmost = rightmost.copy()
    qsecond = second.copy()
    seen.append([qlfmost,qrtmost, qsecond])
    queue.append([qlfmost,qrtmost, qsecond])
    while queue:
        [qlfmost, qrtmost, qsecond] = queue.pop(0)

        unmatched = get_all_unmatched(qlfmost, qrtmost)
        if unmatched:
            # if ENABLE_PRINT: print("The tripple is:")
            # if ENABLE_PRINT: print([qlfmost, qrtmost, qsecond])
            # if ENABLE_PRINT: print("it is unmatched yet!")
```

```
            for person in unmatched:
                if ENABLE_PRINT: print("person is:", person+1)
                #print("before skcycle:",[qlfmost, qrtmost, qsecond])
                first_in_cycle, last_in_cycle, cycle, cursecond = seek_cycle2(preference, qsecond.copy(), person, qlfmost.copy(), qrtmost.copy(), ranking)
                #print("after skcycle:", [qlfmost, qrtmost, qsecond])
                soln_possible, first_in_cycle, last_in_cycle, cursecond, curlfmost, currtmost, cycle = phaseII_reduction2(preference, first_in_cycle, last_in_cycle, cursecond.copy(), qlfmost.copy(), qrtmost.copy(), soln_possible, cycle)
                #print("The tripple is:")
                #print([curlfmost, currtmost, cursecond])
                curtripple = [curlfmost, currtmost, cursecond]
                if not any(all((pref1==pref2).all() for pref1, pref2 in zip(curtripple,tripple)) for tripple in seen) and soln_possible:
                    # if ENABLE_PRINT: print("The new tripple is:")
                    # if ENABLE_PRINT: print([curlfmost, currtmost, cursecond])
                    # if ENABLE_PRINT: print("it is added to the queue")
                    seen.append([curlfmost, currtmost, cursecond])
                    queue.append([curlfmost, currtmost, cursecond])
                #print("after phase ii:", [qlfmost, qrtmost, qsecond])
        else:
            # if ENABLE_PRINT: print("The tripple is:")
            # if ENABLE_PRINT: print([qlfmost, qrtmost, qsecond])
            # if ENABLE_PRINT: print("it is matched already!")
            partner = np.zeros(len(preference[0, :]), dtype=int)
            for person in range(0, len(qlfmost)):
                partner[person] = preference[person, qlfmost[person]]
            if not any(partner.tolist() == p for p in partners):
                partners.append(partner.tolist())

            to_print = friendly_print_sol(partners)


    if ENABLE_PRINT: print("The solution is: ", to_print)
    return partners

```

```python


def gen_random_preference(SIZE = 4):
    preference = np.zeros((SIZE,SIZE), dtype=int)
    for i in range(0,SIZE):
        preference[i,0:SIZE-1]= random.sample([j for j in range(0,SIZE) if j
            != i ],SIZE-1)
        preference[i,SIZE-1] = i
    return preference
```

The preference lists are:

[2 5 4 6 7 8 3 1]

[3 6 1 7 8 5 4 2]

[4 7 2 8 5 6 1 3]

[1 8 3 5 6 7 2 4]

[6 1 8 2 3 4 7 5]

[7 2 5 3 4 1 8 6]

[8 3 6 4 1 2 5 7]

[5 4 7 1 2 3 6 8]

The table after phase-I execution is: 1 — [2 5 4]

2 — [3 6 1]

3 — [4 7 2]

4 — [1 8 3]

5 — [6 1 8]

6 — [7 2 5]

7 — [8 3 6]

8 — [5 4 7]

person is: 1

The rotation exposed is:

E— H S

1— 2 5

8— 5 4

3— 4 7

6— 7 2

The table after rotation elimination is:

1 — [5 4]

2 — [3 6]

3 — [7 2]

4 — [1 8]

5 — [6 1]

6 — [2 5]

7 — [8 3]

8 — [4 7]

person is: 2

The rotation exposed is:

E— H S

2— 3 6

5— 6 1

4— 1 8

7— 8 3

The table after rotation elimination is:

1 — [2 5]

2 — [6 1]

3 — [4 7]

4 — [8 3]

5 — [1 8]

6 — [7 2]

7 — [3 6]

8 — [5 4]

person is: 3

The rotation exposed is:

E— H S

3— 4 7

6— 7 2

1— 2 5

8— 5 4

The table after rotation elimination is:

1 — [5 4]

2 — [3 6]

3 — [7 2]

4 — [1 8]

5 — [6 1]

6 — [2 5]

7 — [8 3]

8 — [4 7]

person is: 4

The rotation exposed is:

E— H S

4— 1 8

7— 8 3

2— 3 6

5— 6 1

The table after rotation elimination is:

1 — [2 5]

2 — [6 1]

3 — [4 7]

4 — [8 3]

5 — [1 8]

6 — [7 2]

7 — [3 6]

8 — [5 4]

person is: 5

The rotation exposed is:

E— H S

5— 6 1

4— 1 8

7— 8 3

2— 3 6

The table after rotation elimination is:

1 — [2 5]

2 — [6 1]

3 — [4 7]

4 — [8 3]

5 — [1 8]

6 — [7 2]

7 — [3 6]

8 — [5 4]

person is: 6

The rotation exposed is:

E— H S

6— 7 2

1— 2 5

8— 5 4

3— 4 7

The table after rotation elimination is:

1 — [5 4]

2 — [3 6]

3 — [7 2]

4 — [1 8]

5 — [6 1]

6 — [2 5]

7 — [8 3]

8 — [4 7]

person is: 7

The rotation exposed is:

E— H S

7— 8 3

2— 3 6

5— 6 1

4— 1 8

The table after rotation elimination is:

1 — [2 5]

2 — [6 1]

3 — [4 7]

4 — [8 3]

5 — [1 8]

6 — [7 2]

7 — [3 6]

8 — [5 4]

person is: 8

The rotation exposed is:

E— H S

8— 5 4

3— 4 7

6— 7 2

1— 2 5

The table after rotation elimination is:

1 — [5 4]

2 — [3 6]

3 — [7 2]

4 — [1 8]

5 — [6 1]

6 — [2 5]

7 — [8 3]

8 — [4 7]

person is: 1

The rotation exposed is:

E— H S

1— 5 4

8— 4 7

3— 7 2

6— 2 5

The table after rotation elimination is: 1 — [4]

2 — [3]

3 — [2]

4 — [1]

5 — [6]

6 — [5]

7 — [8]

8 — [7]

person is: 2

The rotation exposed is:

E— H S

2— 3 6

5— 6 1

4— 1 8

7— 8 3

The table after rotation elimination is: 1 — [5]

2 — [6]

3 — [7]

4 — [8]

5 — [1]

6 — [2]

7 — [3]

8 — [4]

person is: 3

The rotation exposed is:

E— H S

3— 7 2

6— 2 5

1— 5 4

8— 4 7

The table after rotation elimination is: 1 — [4]

2 — [3]

3 — [2]

4 — [1]

5 — [6]

6 — [5]

7 — [8]

8 — [7]

person is: 4

The rotation exposed is:

E— H S

4— 1 8

7— 8 3

2— 3 6

5— 6 1

The table after rotation elimination is: 1 — [5]

2 — [6]

3 — [7]

4 — [8]

5 — [1]

6 — [2]

7 — [3]

8 — [4]

person is: 5

The rotation exposed is:

E— H S

5— 6 1

4— 1 8

7— 8 3

2— 3 6

The table after rotation elimination is: 1 — [5]

2 — [6]

3 — [7]

4 — [8]

5 — [1]

6 — [2]

7 — [3]

8 — [4]

person is: 6

The rotation exposed is:

E— H S

6— 2 5

1— 5 4

8— 4 7

3— 7 2

The table after rotation elimination is: 1 — [4]

2 — [3]

3 — [2]

4 — [1]

5 — [6]

6 — [5]

7 — [8]

8 — [7]

person is: 7

The rotation exposed is:

E— H S

7— 8 3

2— 3 6

5— 6 1

4— 1 8

The table after rotation elimination is: 1 — [5]

2 — [6]

3 — [7]

4 — [8]

5 — [1]

6 — [2]

7 — [3]

8 — [4]

person is: 8

The rotation exposed is:

E— H S

8— 4 7

3— 7 2

6— 2 5

1— 5 4

The table after rotation elimination is: 1 — [4]

2 — [3]

3 — [2]

4 — [1]

5 — [6]

6 — [5]

7 — [8]

8 — [7]

person is: 1

The rotation exposed is:

E— H S

1— 2 5

8— 5 4

3— 4 7

6— 7 2

The table after rotation elimination is: 1 — [5]

2 — [6]

3 — [7]

4 — [8]

5 — [1]

6 — [2]

7 — [3]

8 — [4]

person is: 2

The rotation exposed is:

E— H S

2— 6 1

5— 1 8

4— 8 3

7— 3 6

The table after rotation elimination is: 1 — [2]

2 — [1]

3 — [4]

4 — [3]

5 — [8]

6 — [7]

7 — [6]

8 — [5]

person is: 3

The rotation exposed is:

E— H S

3— 4 7

6— 7 2

1— 2 5

8— 5 4

The table after rotation elimination is: 1 — [5]

2 — [6]

3 — [7]

4 — [8]

5 — [1]

6 — [2]

7 — [3]

8 — [4]

person is: 4

The rotation exposed is:

E— H S

4— 8 3

7— 3 6

2— 6 1

5— 1 8

The table after rotation elimination is: 1 — [2]

2 — [1]

3 — [4]

4 — [3]

5 — [8]

6 — [7]

7 — [6]

8 — [5]

person is: 5

The rotation exposed is:

E— H S

5— 1 8

4— 8 3

7— 3 6

2— 6 1

The table after rotation elimination is: 1 — [2]

2 — [1]

3 — [4]

4 — [3]

5 — [8]

6 — [7]

7 — [6]

8 — [5]

person is: 6

The rotation exposed is:

E— H S

6— 7 2

1— 2 5

8— 5 4

3— 4 7

The table after rotation elimination is: 1 — [5]

2 — [6]

3 — [7]

4 — [8]

5 — [1]

6 — [2]

7 — [3]

8 — [4]

person is: 7

The rotation exposed is:

E— H S

7— 3 6

2— 6 1

5— 1 8

4— 8 3

The table after rotation elimination is: 1 — [2]

2 — [1]

3 — [4]

4 — [3]

5 — [8]

6 — [7]

7 — [6]

8 — [5]

person is: 8

The rotation exposed is:

E— H S

8— 5 4

3— 4 7

6— 7 2

1— 2 5

The table after rotation elimination is: 1 — [5]

2 — [6]

3 — [7]

4 — [8]

5 — [1]

6 — [2]

7 — [3]

8 — [4]

The solution is: [[(1, 4), (2, 3), (5, 6), (7, 8)], [(1, 5), (2, 6), (3, 7), (4, 8)], [(1, 2), (3, 4), (5, 8), (6, 7)]]

[[3, 2, 1, 0, 5, 4, 7, 6], [4, 5, 6, 7, 0, 1, 2, 3], [1, 0, 3, 2, 7, 6, 5, 4]]