Design and Analysis of Algorithms
CSCI 5454

Deepak Shivarudrappa
Jesukumar Ravikumar

STABLE MATCHING

1 Introduction:

Sometimes, we would like to pair up elements of a set in that their individual preferences are taken into account and the resulting pairing / matching is stable. To visualize this scenario consider the following example; Job seeking students in their final year of college apply to open positions in various companies, and each student will have his/her own preference list for the companies applying to. Similarly all the companies will have a preference list of the prospective employees. Let each student $s_i \varepsilon S$ (set of students) have a preference list sp_i of the companies applied to, and each company $c_i \varepsilon C$ (set of companies) have a preference list cp_i of the students. We would now like to find a suitable one-one matching pair between the set of elements in *S* and *C* such that the resulting matching set is **perfect**[*] and **stable**[*].

STUDENT	PREFERENCE LIST
Eric	Google, Amazon, Apple, (Microsoft, Yahoo)
Donna	Apple, Google, Microsoft, Amazon, Yahoo
Fez	Yahoo, (Google, Amazon), Microsoft, Apple
Steven	Amazon, Apple, Microsoft, yahoo, Google
Kelso	Microsoft, Apple, yahoo, Google, Amazon
Jackie	Google, Microsoft, Apple, Yahoo, Amazon

COMPANY	PREFERENCE LIST
Google	Eric , Jackie, Steven, Fez, Kelso
Yahoo	Fez, Eric, Jackie, Kelso, Steven
Amazon	Donna, Eric, Kelso, Fez, Steven
Microsoft	Steven, Donna, Fez, Jackie, Eric
Apple	Jackie , Kelso, Eric, Donna, Fez

The above listed sets can be matched in many ways; some of them are bad and some of them good.

Consider the following matching:

Steven --- Google Eric --- Yahoo Donna --- Amazon
Fez --- Microsoft Jackie --- Apple

This is an example of a bad matching, if observed carefully, Google prefers Eric over Steven and Eric prefers Google over yahoo. This means, there is nothing stopping from Google hiring Eric and Eric quitting yahoo and joining Google. Therefore there is some **Instability** in the matched set. Now consider this set:

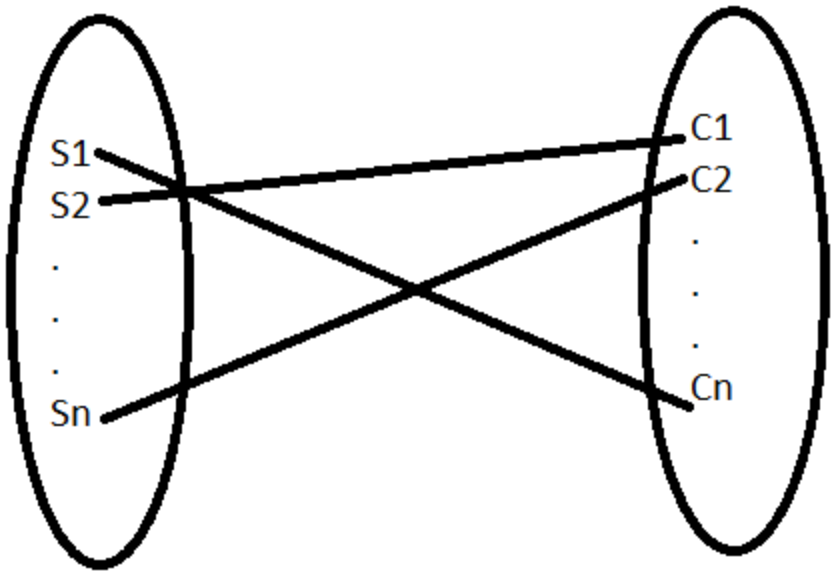
Eric --- Google	Donna --- Amazon	Fez --- yahoo
Steven --- Microsoft	Jackie --- Apple	

The self-interest of every individual and employer prevents them from abandoning their current matching. The current pairings are such that hiring people behind the scene (like the instance of Google and Eric coming together in the last example) is not possible. Thus this matching has some inherent stability to it. But how do we arrive at this matching from amongst all the different possibilities? The algorithms that deal and solve such problems are the Stable matching Algorithms. The following text presents two such algorithms with some variations;

1. The Stable marriage problem
2. The Stable roommate problem.

These algorithms have found their application in matching graduate applicants to schools and matching residents to hospitals.

In general what we are trying to do is to find a matching pair between the two sets which is **stable and all the elements of the final pairing are satisfied.**



Before going ahead, we will simplify the problem a little and come up with a more general problem statement. We will consider the cardinality of the sets to be equal, and will consider only a strict order of preferences. Applying this simplification to the above example we have n elements each in set S and C, each student ranks all the companies and cannot be indifferent to two different companies, similarly the companies rank the employees.

The above problem is analogous to the Stable marriage problem where we try and match a set of n men ε M $\{m_1.....m_n\}$ and n women ε W $\{ w_1.....w_n\}$ taking into account their strict order of preference.

2. THE STABLE MARRIAGE PROBLEM

2.1 Problem Statement:

Given 'n' men and 'n' women, find a "suitable" matching between men and women.
Participants rank every member of opposite sex. Each man lists women in order of preference from best to worst. Each woman lists men in order of preference.

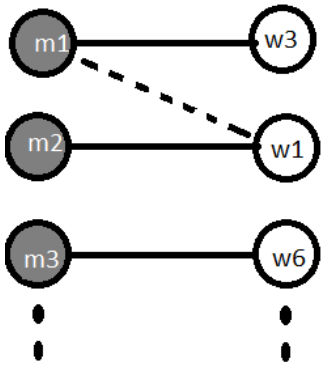
Let **M** denote the set of all men $\{m_1.....m_n\}$
Let **W** denote the set of all women $\{w_1.....w_n\}$
Let **Z** denote the set of all possible pairs in M x W (there are n^2 possible pairs considering n elements in each set)
Let **S** denote a stable set in **Z**.

2.2 Definitions

We will take a closer look at what stability is and define some more terms before we continue.

Instability

We say a set of pairs is unstable if there is a pair which prefers to be matched together than be with their current assigned partners.



Consider the pairs (m_1,w_3), (m_2,w_1) in a matched set **A** as shown in the figure above, now suppose m_1 ranks w_1 higher than w_3 and w_1 ranks m_1 higher than m_2, then m_1 and w_1 would want to be together and may abandon their current partners w_3 and m_2. This is a case of instability with respect to (m_1, w_1).

Perfect matching

A set in which all the elements are paired, and none of the elements appear in more than one pair is called a perfect matching.

Stability

A matching is said to be stable if it is a perfect matching with not instabilities in it. To put it formally, stability exist in S, if there is no pairs (m_1, w_1) and (m_2, w_2) such that

- m_1 prefers w_2 over w_1 and
- w_2 prefers m_1 over m_2

2.3 PROPOSE REJECT ALGORITHM [GALE-SHAPLEY, 1962]

"Men Propose, Women Dispose"

2.3.1 ALGORITHM

Let *'m'* be a randomly selected man from the set of free men. *'m'* proposes to the highest ranked woman in his preference list to whom he has not yet proposed to. The woman *'w'* if free will enter into an engagement with *'m'*. If the woman was already engaged, she will then decide on rejecting *m's* proposal or accepting m's proposal and rejecting her current partner based on relative ranking for m and the current partner in her preference list. If m is ranked higher, *w* will accept *m's* proposal, rejecting her current partner, else she will reject *m's* proposal. This process is repeated until no man if free or a Stable match is found.

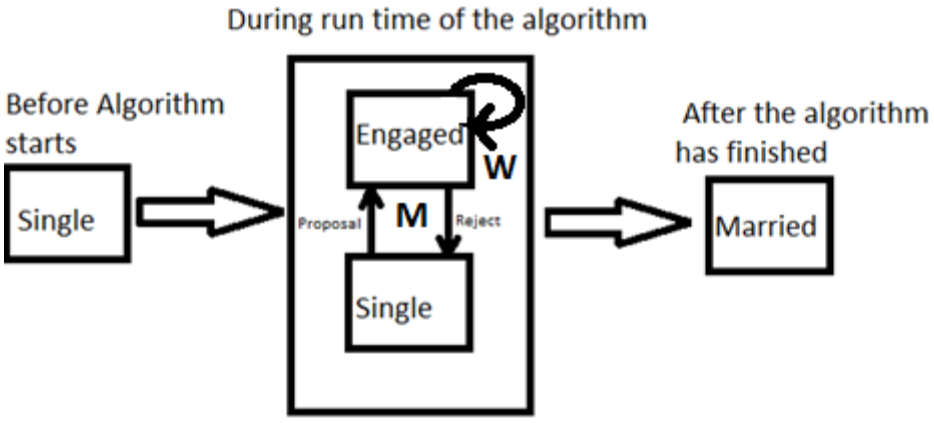


Illustration of states of men and women during the course of algorithm

Example to show how the algorithm works :

Men's preference list	Women's preference list
m_1 -> w_1 w_2 w_3	w_1 -> m_2 m_1 m_3
m_2 -> w_1 w_3 w_2	w_2 -> m_3 m_2 m_1
m_3 -> w_2 w_1 w_3	w_3 -> m_1 m_3 m_2

m_1 proposes to w_1, since w_1 is single the proposal is accepted

m_2 proposes to w_1, since m_2 is ranked higher than m_1, m_1 is rejected by w_1

m_1 is single now

m_1 proposes to w_2 , proposal accepted

m_3 proposes to w_2, since m_3 ranks higher than m_1, m_1 is again rejected by w_2

finally, m_1 proposes to w_3 and is accepted.

```
// INTILIZATION
  for i=1 to n
  {
    push(STACK,m[i])                          //push the set of men M{m1...mn} onto stack
    m[i].status = w[i].status = single;       //Initialize the status to 0
  }

  while(STACK NOT EMPTY)                       // Check if any man is free
{
  x_man=pop();                                // pop from stack
  if(x_man.status = single)                   // Check if man is single
  {
     do{                                      //Loop until man is engaged
         woman=preference(x_man)              //Extract the current highest ranked woman from the preference list

            if(propose_to(woman)=SUCCESS)     //Check if the proposal is accepted, and on if successful
            {
                if(woman.status=engaged)      //If the woman is engaged, she rejected the person she was engaged to
                 {
                    man_rejected=woman.engaged_to
                    push(STACK,man_rejected) //push the rejected man on to the stack
                 }

                woman.status = engaged        //update the status and the person each engaged to
                x_man.status = engaged
                woman.engaged_to= x_man
                x_man.engaed_to = woman
            }
```

```
            else
            {
               update_preference(x_man)      //Update the preference
            }
    }while(x_man.status=single)
  }
}


return
```

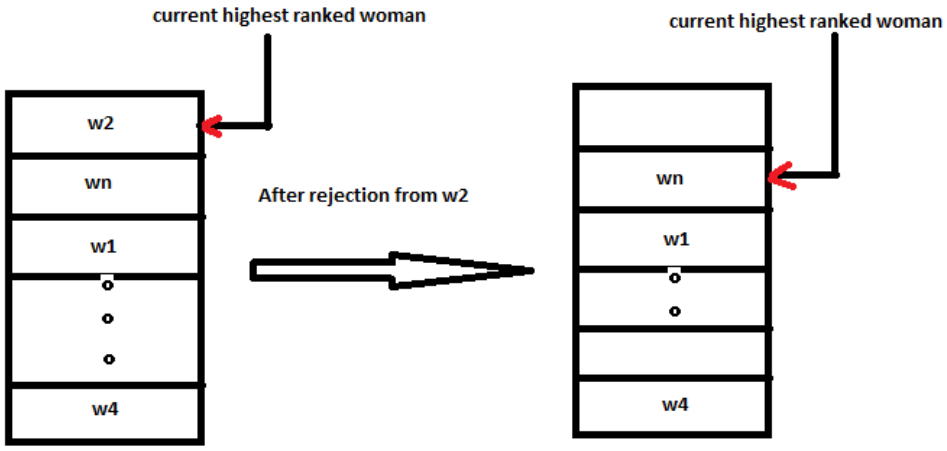
Pseudo Code for Propose Reject algorithm

2.3.2 Implementation:

The algorithm has an upper bound of n^2. So it is very important that the running time of *while loop* is minimum, preferably constant running time. Some operations performed in the loop and data structures used to implement them so that we have constant running time for the loop is given below

- Identify free men: *Set, Stack*
- Highest ranked women that man has not proposed to yet:

We can use a stack for each man. The stack pointer will point to the highest ranked woman in his list and if he gets a reject from a woman, that person will be popped out of the stack.



Men's preference list implementation

- Identify a woman's partner: *list of n elements with women as indexes*
- Check and compare the rankings for current partner and proposer for women 'w':

  n X n Rank matrix

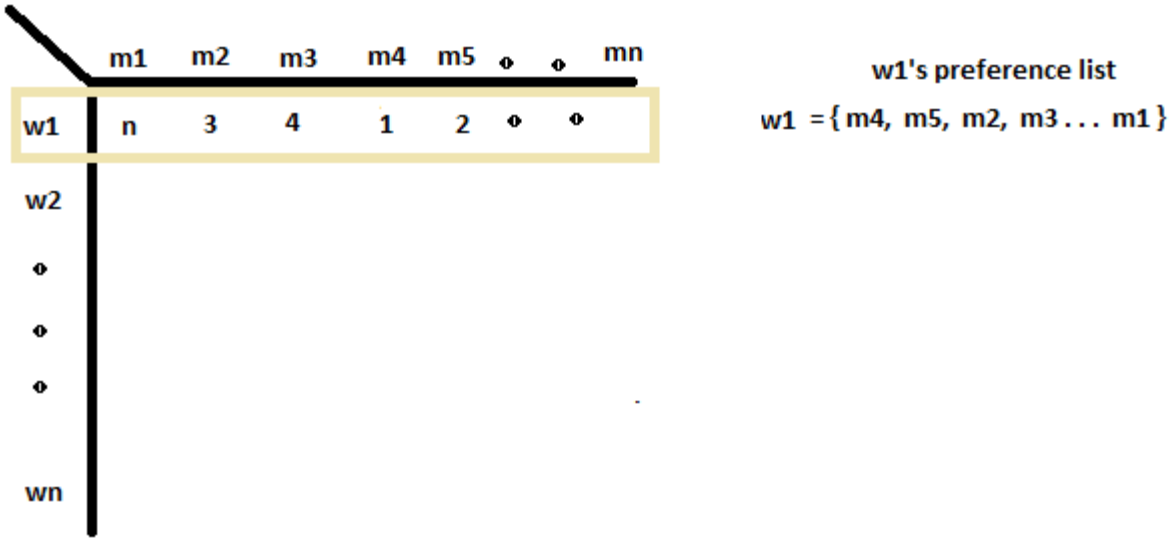


Figure showing implementation of Rank matrix

Observe how w_1's preference list is stored in the matrix, since m_4 is the first choice of w_1, there is "1" in (row 1, col 4). Similarly, the values are stored for others.

Any A_{ij} = Rank of man m_j in the list of w_i

To obtain the rank of a man in the list of w1 we would index into the matrix

→ Matrix[w1][mi] where i identifies the man.

Using the above data structures we can keep the running time bounded by $\boldsymbol{n^2}$.

2.3.3 Analysis of Algorithm:

In this section we will analyze the results of the algorithm in terms of their correctness and performance. To start with we state some properties observed in the algorithms:

If a woman w_1 is engaged at time t_1, then she will remain engaged from t_1 till the termination of the algorithm and the partner she pairs with cannot get worse with time.

If (m_1, w_1) are engaged and another man m_2 proposes to w_1, w_1 may accept the proposal or reject it in favor of m_1. In either case the woman w_1 continues to be engaged. An engaged woman w_1 will accept a different proposal only if the m_2 is ranked higher on her list. Hence every time a woman breaks an engagement she gets engaged to a better partner. The above does not hold well from a man's point of view. ***The ranking of the women he proposes to, worsens with time.***

No man m_1 or woman w_1 can remain unpaired at the termination of algorithm.

Consider a situation where a man m_i is free at the termination of algorithm. This means that man m_i has already proposed to all women w ε W, else the *while* loops would not terminate. But for all *'n'* women to reject m_1's proposal, they must already be paired better ranked men. But this is not possible since we have only *'n'* men in M. Thus no man m_1 can remain free on termination of the algorithm. To prove the latter, we assume a situation where woman w_1 remains unpaired after termination. For this to happen none of the men should have proposed to w_1. This implies that all *'n'* men should have been paired already. But this is not possible since we have only *'n'* women. Thus w_1 cannot be unpaired at termination. A corollary of the above statement would be; ***A man cannot be free after having proposed to all the women.***

The pairs returned by the algorithm is both perfect and stable

Applying the earlier statement to all the men, we can safely say that, in worst case all men would have proposed to all women and be paired at the end. Thus the matching returned is *perfect.*

The stability of returned perfect matching S can be proved by assuming instability and contradicting it. Assume (m_1, w_1) and (m_2, w_2) belong to a stable matching S. For instability to exist in S,

- m_2 must prefer w_1 to w_2
- w_1 must prefer m_2 to m_1

Since m_2 is paired with w_2 in S, by definition the last women m_2 proposed to is w_2.

Assume condition **m_2 must prefer w_1 to w_2** (1) to be true:
Then m_2 must have proposed to w_1, and m_2 was rejected or engaged to and later rejected by w_1 in favor of a preferred partner m_3.

$$\mathbf{m_2 > m_3\ (w_1)}$$

Then w_1's final partner in S, m_1 must have a rank same as or equal to m_3.

$$\mathbf{m_1 \geq m_3\ (w_1)}$$

Thus w_1 prefers m_1 over m_2 which contradicts condition **w_1 must prefer m_2 to m_1** (2).

Assume condition **w_1 must prefer m_2 to m_1** (2) to be true:
Since w_1 is paired with m_1 at termination, she must have not received a proposal from m_2. And since m_2 is paired w_2, this contradicts condition **m_2 must prefer w_1 to w_2** (2).
Since both the conditions cannot be satisfied at the same time we can say that instability cannot exist in S. Thus **S is Stable**

2.3.4 Choices and its effect on result:

Although our pseudo code implementation has a ordered structure to choosing free man for the next proposal, according to original G-S algorithm this is constrained by the only condition that the man should be free. Otherwise any man can be chosen from the set of free men to make the next proposal. This does not affect the final stable matching S.

Any order of execution will return a set S* which is set of (m, best_valid(m)) for all men.

Valid partner: A partner 'w' is called a valid partner of 'm' if the pair (m, w) appears in any of the possible stable matching sets S.
Best valid partner: The highest ranked valid partner of 'm' / 'w' according to his/her preference list.

Consider a man m_1 with set of valid partners $\{w_1, w_2, \ldots\}$. Assume an order of proposals (execution) E that return S. Let the first rejection involve m_1, w_1. w_1 rejects m_1's proposal in favor of highly ranked m_2.
Since the w_1 is a valid partner of m_1, there exists a stable matching S" where m_1, w_1 are paired. In S", let m_2 be paired with woman w_2 other than w_1. For stability to exist m_2 must prefer w_2 over w_1. If this was the case then m_2 should have proposed to and rejected by w_2 before being paired with w_1 in E. But m_1's rejection by w_1 is the first rejection and hence there cannot be any w_2 such that m_2 prefers w_2 over w_1. Therefore set S" is unstable with respect to pair (m_2, w_1). Thus there cannot be any perfect matching that is table and has the pair m_1, w_1. Therefore w_1 is not a valid partner of m_1. All women whom m_1 proposes to and is rejected by cannot belong to his set of valid partners. The woman w_2 who accepts m_1's proposal belongs to set of valid partners since S is stable and m_1, w_2 belong to S. Thereby proving all men always end up with their best valid partner.

Also to note here is the fact that if m_1 is rejected by w_1, then there cannot be a matching S" that has the pair m_1, w_1 and is also stable. We will use this fact to bound the running time later.

2.3.5 Unfairness Phenomenon:

The Propose and Reject algorithms has the property of being unfair to those set of individuals who make the decision of accepting or rejecting proposals i.e. the women in the stable marriage problem. Consider a case where no two men have the same women as their most preferred partner. In such a case the algorithms outputs the same stable matching independent of the choices/ preferences made by women. There is a probability that all women get their least preferred partner but still the matching set S is stable. Hence this algorithm is called *men optimal or women pessimal* algorithm. This can be reversed i.e. made *women optimal and men pessimal* by interchanging the roles of men and women. To put it in generally propose reject

algorithm always yields a proposer optimal result.

Always choose to propose if following a propose reject algorithms.

Now having observed the two extremes, the question arises, can a stable matching that is a compromise of both these be possible? Yes that is possible by tweaking the G-S algorithm. Instead of forcing a woman to accept any proposal when she is free, if we allow her to reject a proposal when she is free if the proposer is in lower half of the preference list. And after the first round of proposals we will have a minimum of half of 'n' final pairs have been formed. We eliminate those from the list and repeat the above process or apply a unmodified G-S algorithms at this stage to get a solution S different from both man and woman optimal solutions. Note the limit chosen i.e. half is arbitrary. If we choose this to lower n-1 elements then the algorithm would return a woman optimal solution even with man making proposals.

Example:

Consider the preference lists as below

m_1	w_1	w_2	w_3	w_4
m_2	w_2	w_3	w_4	w_1
m_3	w_3	w_4	w_1	w_2
m_4	w_4	w_1	w_2	w_3

w_1	m_2	m_3	m_4	m_1
w_2	m_3	m_4	m_1	m_2
w_3	m_4	m_1	m_2	m_3
w_4	m_1	m_2	m_3	m_4

The solutions using different algorithms are as shown below:

Men optimal
$m_1 – w_1$
$m_2 – w_2$
$m_3 – w_3$
$m_4 – w_4$

Women optimal
$m_1 – w_4$
$m_2 – w_1$
$m_3 – w_2$
$m_4 – w_3$

Intermediate
$m_1 – w_3$
$m_2 – w_4$
$m_3 – w_1$
$m_4 – w_2$

2.3.6 Bound on the running time:

The bound on the running time is given by O($\boldsymbol{n^2}$) . The number of proposals/ iterations of *while* is used to bound the running time. Now having a look at the set of men and women we can infer that a maximum of $\boldsymbol{n^2}$ proposals can be made. But we still do not know if a man m_1 can propose to same woman w_1 twice? Note in section 2.3.4 we proved that
If m_1 is rejected by w_1, then there cannot be a matching S" that has the pair m_1, w_1 and is also stable.

Therefore we can safely say that there is no use proposing twice to a same woman. The result will always be the same. Thus the while loop can be executed at most $\boldsymbol{n^2}$ times. The factors such as number of free man, number of engaged pairs do not necessarily increase with each iteration and hence are not used to bound the running time.
An example that gives worst case running time is when all men have same preference list and all women have the same preference list. Such a input cause $\sum_1^n\ i$ proposals to be made which is O($\boldsymbol{n^2}$).
The best case running time is O($\boldsymbol{n^1}$)**.** This is possible under the condition that all *'n'* men prefer different women for their top spot. In such a case everybody proposes once and a stable matching is found.

2.4 Real World Example: National Resident Matching Program

NRMP is a non-profit organization which used the stable matching algorithm to post residents to hospitals in US. Their algorithms used the set of Hospitals H and set of residents R, and their order of preference to decide on the matching. The similarities and differences as compared to stable marriage problems are list below

(i) Set of Hospitals are analogous to set of men M responsible for proposing pairings.
(ii) Set of residents R is analogous to set of women W, who make decision on accepting or rejecting proposals
(iii) Unlike the monogamy policy followed by the stable marriage algorithm this follows polygamy. Here more than 1 resident can be matched to a particular hospital. However the number of residents matched to a particular hospital is limited by the requirement at the hospital
(iv) All residents need not ranks all the hospitals in H. They can declare some h as undesirable
(v) If the number of residents is more than the number of vacancies of all the hospitals combined, some of the residents would remain unmatched.
(vi) Similarly vacancies in some hospitals may not be filled because of shortage of residents.
(vii) This algorithms was redesigned later to make it resident optimal and accommodate couples.

The condition to be satisfied for stability here is there is no R, H pair such that

- Resident R prefers hospital H, to his current posting and
- Hospital H has a vacant slot or prefers R over one of its residents

Pseudo Code:

```
While there is a hospital h who has an empty slot (i.e. vacancies[h] ≠ 0) and
        has not offered position to every resident
{
        Choose a hospital h;
        Offer vacancies[h] positions to highest ranked residents R_h to whom h has not offered;

        For every resident in R_h
        {
                If r is free then
                        r accepts offer made by h
                        vacancies[h]--;
                Else r is currently accepted offer from h'
                {
                        If r prefers h' to h then
                                Vacancy in h remain unfilled;
                        Else If r prefers h to h'
                                r accepts offer made by h;
                                vacancies[h]--;
                                vacancies[h']++;
                        Endif
                }
        }
}
```

Return the set S of pairs

3. STABLE ROOMMATE PROBLEM

So far, we have seen how to pair up elements of two different set i.e. the set we considered for the stable marriage problem could be represented as a bipartite graph. Sometimes, it may be necessary to pair up elements within a single set. Think of the case where we have n students who are seeking a roommate from a set of n students of which they are a part of. So, the next algorithm looks at this sort of problem.

3.1 Problem Statement:

Given 'n' students, where n is even, find a "suitable" matching. Participants rank all the other members in the group. The ordering in the preference list is a strict order.

As in the case of the stable marriage problem, each student in the set ranks all the other n-1 students in a strict order of preference. The algorithm then tries to pair up the elements within this set considering their preferences and may come up with a stable matching.

Unlike the Stable marriage problem it is not guaranteed that the stable roommate algorithm will always result in a stable matching. This fact can be illustrated with an example.

Example:

STUDENT	PREFERENCES
Howard	Leonard, Kutapalli, Sheldon
Leonard	Kutrapalli,Howard, Sheldon
Kutrapalli	Howard, Leonard , Sheldon
Sheldon	Leonard, Kutrapalli, Howard

Pairing up the students with the above preference list will always result in a matching that has instability. Any pair with Sheldon is unstable.

Kutrapalli----- Howard
Leonard ----- Sheldon
In this matching, Leonard---Sheldon pair is unstable.

3.2 Algorithm

The algorithm for the stable roommate problem is divided into 2 phases.

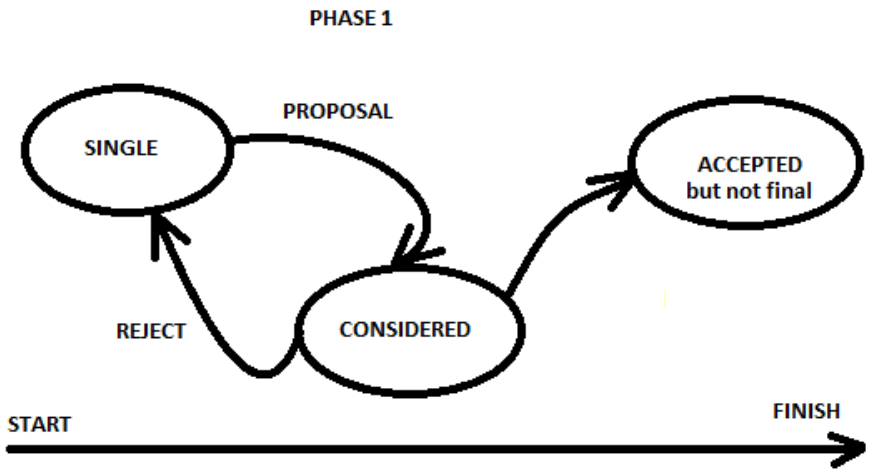
The first phase is very similar to the stable marriage problem with some variation. The result of phase 1 may not always be a stable matching, the algorithm then proceeds to the second phase. The second phase readjusts the matching and tries its best to obtain a stable matching.

3.2.1 Phase 1:

Proposal:
Each student S_i proposes to the highest ranked student on his list. If he is rejected he continues to propose to the next best in his list.
Rejection/Consideration:
Each student S_i on getting a proposal will consider it if that is the only proposal or is the best one that he has got yet and may reject the other one he is holding.

To illustrate this stage let us consider an example:

STUDENT	PREFERENCE LIST
S_1	S_2 S_6 S_4 S_3 S_5
S_2	S_4 S_5 S_6 S_1 S_3
S_3	S_6 S_1 S_2 S_4 S_5
S_4	S_5 S_1 S_3 S_6 S_4
S_5	S_3 S_4 S_1 S_2 S_6
S_6	S_2 S_3 S_1 S_5 S_4



S_1 --------> S_2
S_2 --------> S_4
S_3 --------> S_6
S_4 --------> S_5
S_5 --------> S_3
S_6 --------> S_2 , S_2 rejects S_1 since S_6 is ranked higher
S_1 -------->S_6 , S_6 rejects S_3
S_3 --------> S_1

For a stable matching, if there is a pair (a,b) there should also be a pair (b,a).The matching after phase 1 depending on whether stable or not enters the second phase.

```
//Initialize
 for(i=0;i<n;i++)
    received_proposal[i]=FALSE                      // Initialize the array to zero

 for student in 1 to n do                           // Loop for each student
  {
    proposer = student
    do{                                              // Loop until the next choice of the proposer is in the set received_proposal
        temp = preference(proposer)
        if(propose_to(temp)= SUCCESS)                // If accepted
        {
            if (received_proposal[temp]=TRUE)        // If temp has already received proposal, reject the current one
                {   temp.consider = proposer         // The proposal is considered
                    update_preference(proposer,temp) // Track all those to whom proposal has has been made
                    proposer = temp.rejected         // The new proposer is the rejected one
                }
        }
      }while (received_proposal[temp]= TRUE);
    received_proposal[temp]=TRUE                     // record that temp has received a proposal

  }
```

Pseudo code for phase 1 of Stable Roommates algorithm

3.2.2 PHASE 2:

Phase 2 is a little complex, after finishing off with round 1, each student has proposed to certain students and hasn't proposed to some of them. From the pseudo code above it is seen that a student doesn't propose more than once to the same person. And we keep track of the people proposed to by updating the preferences in the update_preference() function. Therefore, after phase 1 the preference list of each student is in a reduced form and doesn't contain all the initial elements.

Phase 2 now tries to shrink this reduced preference list further by eliminating certain choices and tries to slowly steer the set to a stable direction.
In this step, a cycle is identified (c_1.....c_r) and the elements in the identified cycle are used to reduce the list.

For a particular student P_i, his next choice in the list is identified as Q_i. P_{i+1} is the last choice in Q_i's preference list and Q_{i+1} is the next choice in P_{i+1}'s list. A subset of P_i identified in this set can be mapped to the circle or a rotation (c_1.....c_r) such that $c_{r+1}=c_1$. Once such a rotation is identified, further reduction of the preference list is possible. By forcing c_1 to propose to its next choice leads to a chain of proposals and rejects. If c_1 proposes to its next choice, then c_2 is rejected by c_1's next choice. This means, c_2 will now propose to its next choice and so on. This in a way tries to work incrementally towards a stable solution.

Nothing is guaranteed, it may so happen that a student s_i may end up being rejected by all the members of the group in which case there is no stable matching. The second phase of the

algorithm terminates only when the set is a stable one or if a certain person has been rejected by everyone.

Phase 2 in general
While(a preference list is >0)
{
    Locate_cycle()
    Apply the steps discussed above
}

3.3 Analysis of Algorithm:

Lemma 1: If at any stage of the algorithm, p_1 rejects p_2's proposal, then p_1, p_2 can never be a pair in stable matching.

Let the above rejection be the first rejection in our order of execution. In our actual case above p_1 rejects p_2 in favor of p_3. Assume a stable matching S where p_1, p_2 are partners. For stability to exist in S, p_3 should have a partner p_4, who is ranked higher on his list than p_1. Now before p_3 could propose to p_1, he must have been rejected by p_4 and this must have preceded p_1's rejection of p_2. This contradicts fact that rejection of p_2 was the first rejection and hence cannot be possible. Some corollaries that can be deduced are

Corollary: If p_1 proposes to p_2 then, in a stable matching S,

- p_1 cannot have a partner better than p_2
- p_2 cannot have a worse partner than p_1

Since p_1 proposes to p_2, this means all other p_x whom p_1 prefers over p_2 have rejected him and therefore by *Lemma 1*, they cannot be partners and p_2 is the best available partner. To prove second claim let's assume a stable matching S which has a pairing p_2, p_3, where p_2 prefers p_1 over p_3. Now there is an instable pair p_1, p_2 and contradicts our claim making S impossible.

Corollary: If a person p_x is rejected everybody else no stable matching can exist. By *Lemma 1* p_x cannot be anybody's partner and hence matching got will not be perfect or stable.

Corollary: On termination of first phase of the algorithm, we can reduce the preference list of individuals by applying corollary. If p_2 holds proposal from p_1, all preferences ranked higher than p_2 in p_1's list can be deleted and all preferences below p_1 in p_2's list can be deleted. Person p_1 will be the last one on the list of p_2 and p_2 will be the first entry in p_1's list. Person's p_1 and p_2 both appear on each other's list.

Lemma 2: If in the reduced lists, all lists contain a single element, then list specifies a stable matching

Person p_1 has p_2 on his list and therefore p_2 has p_1 on his list and the first entry in the reduced list is the best partner that a person can have. Hence there will be no instability in pair p_1, p_2.

Lemma 3: In all or nothing set r_1, r_2... r_n, if b_i denotes the first preference of each r_i, then

- ***In any stable matching for the reduced list given, (r_i, b_i) are partners for all values of i or for no value of i.***
- ***If a_i, b_i are partners in any stable matching then there is another stable matching in which they are not.***

b_i is first on a_i's list implies a_i is last on b_i's list. Also since b_i was found in a_{i-1}, b_i will prefer a_{i-1} to a_i. For stability a_{i-1} must be paired with some whom he prefers over b_i. We know that b_i is second preference and only preference of a_{i-1} has over b_i is b_{i-1}. Repeating this a_i and b_i must be partners for all values of *'i'* for stability.

Let S' be a matching of a_i, b_{i+1}. In the previous case all a_i got their best possible partners and all b_i got their worst. So in case of a stable matching S' where a_i and b_i are not partners, any instability that exists will involve a_i. We will now claim that S' is stable and prove.

Let us say a_i prefers x over b_{i+1} in the original preference list. Then

- ***x is b_i:*** In which case x prefers new partner a_{i-1} to a_i
- ***x is preferred over b_i:*** But x is not in the reduced preference list of a_i, therefore x must have rejected a_i's proposal earlier or must have been forced to reject a_i during earlier iterations of phase 2. In either case x prefers its partner in S' over a_i
- ***b_i is preferred over x:*** In this case since x is between b_i and b_{i+1}. Since x is not in the reduced preference list, x must have rejected earlier in phase 1, making x prefer its partner in S' over a_i.

The running time of the stable roommates algorithms is bounded by $O(\boldsymbol{n^2})$.

References:

- ***"Algorithm Design"*** by *Jon Klienberg* and *Eva Tardos*
- *D. Gale and L. S. Shapley*: ***"College Admissions and the Stability of Marriage"*** [1962]
- *Robert W. Irving*: ***An Efficient Algorithm for the "Stable Roommates" problem*** [1984]