Stable Marriages

The problem

Input: $n$ men, $n$ women, each with their own preference list of partners of the opposite sex

\[
\begin{array}{ccc}
 & m_1 & w_1 \\
 & m_2 & w_2 \\
 & m_3 & w_3 \\
 & m_4 & w_4 \\
 & m_5 & w_5 \\
\end{array}
\]

Goal: Match up men and women in pairs so that each man and each woman is married to a partner of the opposite sex and all marriages are stable.

Two marriages $(m, w)$ and $(m', w')$ are unstable if $m$ prefers $w'$ over $w$ and $w'$ prefers $m$ over $m'$.

The proposal algorithm

\[
\begin{align*}
\text{while there is an unmarried man } m \\
\text{do } & m \text{ proposes to the next woman } w \text{ in his preference list} \\
\text{if } & w \text{ is unmarried or prefers } m \text{ over her current partner } m' \\
\text{then } & w \text{ divorces } m' \\
\text{end if} & w \text{ marries } m
\end{align*}
\]
Questions to ask about the (any) algorithm

1. Is there always a solution to the problem we're trying to solve?

2. Does the algorithm always find a solution, i.e., is the algorithm correct?

3. Does the algorithm always terminate?

4. How quickly does the algorithm terminate?

A modified proposal algorithm

while there is an unmarried man \( m \) who has not proposed to all women yet

do
\( m \) proposes to the next woman \( w \) in his preference list
if \( w \) is unmarried or prefers \( m \) over her current partner \( m' \)
then \( w \) divorces \( m' \)
\( w \) marries \( m \)

Lemma: The modified proposal algorithm terminates after at most \( n^2 \) iterations.

Proof: 
- There are \( n \) men.
- Each can propose to \( n \) women.
- In each iteration one proposal is made.
- No man proposes to the same woman twice.

\[\square\]
Lemma: When the modified proposal algorithm terminates, every woman (and hence every man) is married.

Proof: Assume the contrary.

⇒ ∃ man m and woman w that are unmarried when the algorithm terminates.

Since m is unmarried and the algorithm terminates, m must have exhausted his preference list.

⇒ m must have proposed to w at some point.

w is married after the proposal (not necessarily to m).

A woman, once married, stays married (not necessarily to the same man).

⇒ w must be married when the algorithm terminates, a contradiction.

□

Corollary: The standard proposal algorithm terminates after at most $n^2$ iterations.

Proof: We proved that the modified algorithm terminates after $\leq n^2$ iterations. By the previous lemma, that algorithm never uses its ability to terminate because all preference lists are exhausted. Thus, it behaves the same as the original algorithm.

⇒ The original algorithm terminates after $\leq n^2$ iterations.

□
Lemma: The set of marriages obtained at the end of the algorithm is stable.

Proof: Assume the contrary.

\[ \Rightarrow \exists \text{ marriages } (m, w) \text{ and } (m', w') \text{ s.t.} \]

\[ m \text{ prefers } w' \text{ over } w \text{ and } w' \text{ prefers } m \text{ over } m' \]

\[ \Rightarrow m \text{ must have proposed to } w' \text{ before marrying } w \]

\[ \Rightarrow w' \text{ likes the man } m'' \text{ she is married to at least as much as } m \]

( either \( m'' = m \) or \( w' \) rejected \( m \) in favor of \( m'' \)).

Let \( m'' = m_1, m_2, \ldots, m_t = m' \) be the sequence of partners \( w' \) has from this moment forward.

If \( t = 1 \), then \( m' = m'' \). Since \( w' \) likes \( m'' \) at least as much as \( m \) and \( m' \neq m'' \), \( w' \) likes \( m'' = m' \) strictly better than \( m \), a contradiction.

If \( t > 1 \), then \( w' \) likes \( m_{t+1} \) strictly better than \( m_t \) for all \( 1 \leq i < t \) because she divorces \( m_i \) for \( m_{i+1} \).

Thus, she likes \( m' = m_t \) strictly better than \( m'' = m_t \), who she likes at least as much as \( m \).

\[ \Rightarrow w' \text{ likes } m' \text{ strictly better than } m, \text{ again a contradiction}. \]
More interesting questions

1. Does the solution depend on the order in which the men propose?  No!!!

2. Is the solution equally fair to both sexes?
   The men fare much better:

   \[ w_1 \quad \quad m_1 \quad w_1 \quad \quad w_2 \quad m_2 \quad w_2 \quad \quad w_3 \quad m_3 \quad w_3 \quad \quad w_4 \quad m_4 \quad w_4 \quad \quad w_5 \quad m_5 \quad w_5 \quad \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \]

3. Can we design a faster algorithm?
   Yes, using randomization.

4. How fast can we implement this algorithm?
   (Next topic)