CompSci 260P

Fundamentals of Algorithms with Applications

Fall 2017

- Lecture: Tu Th 3:30-4:50pm SH 174

  no lecture on Thursdays October 5, October 12
- Discussion: Tu 7:00-7:50pm SH 174

- Course Web Page

  `http://www.ics.uci.edu/~dan/class/260P/`
- Course Notes: (password protected)

  `http://www.ics.uci.edu/~dan/class/260P/notes/`
  - user name is your UCI ID (ALL CAPS)
  - password is your student number

Reading, Homework, etc.

- Reading
  - read the relevant book sections and handouts **before** my lecture
  - I will lecture on only **some** of the material

    (there is not enough time for me to talk about every page in the book)
  - you are responsible for **all** of the material

Reading, Homework, etc.

- Reading
  - read the relevant book sections and handouts **before** my lecture
  - I will lecture on only **some** of the material

    (there is not enough time for me to talk about every page in the book)
  - you are responsible for **all** of the material

- Homework
  - submit using EEE dropbox (see `https://eee.uci.edu/help/dropbox/students/`)
  - **no late submissions will be accepted**
  - submissions graded on effort, not on correctness
  - some (but not all) solutions will be discussed in class
  - I may call on one of the students who has solved a problem to present the solution to the class

Reading, Homework, etc.

- Reading
  - read the relevant book sections and handouts **before** my lecture
  - I will lecture on only **some** of the material

    (there is not enough time for me to talk about every page in the book)
  - you are responsible for **all** of the material

- Homework
  - submit using EEE dropbox (see `https://eee.uci.edu/help/dropbox/students/`)
  - **no late submissions will be accepted**
  - submissions graded on effort, not on correctness
  - some (but not all) solutions will be discussed in class
  - I may call on one of the students who has solved a problem to present the solution to the class

- Solved problems in the textbook
  - will not be assigned, but you are encouraged to solve them
  - you will find these more useful if you solve them before reading the solutions

Textbooks / Recommended Reading

Textbook:

- Kleinberg and Tardos, *Algorithm Design*, Addison Wesley, 2006

Other books:

- Ahuja, Magnanti and Orlin, *Network Flows*, Prentice Hall, 1993
- Cormen, Leiserson, Rivest, and Stein,
  Introduction to Algorithms, MIT Press (third edition), 2009
- Garey and Johnson, *Computers and Intractability:*
  A Guide to the Theory of NP-completeness,
  W. H. Freeman and Company 1979
- Goodrich and Tamassia,
  Algorithm Design and Applications, Wiley, 2014
- Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994
- Dasgupta, Papadimitriou and Vazirani,
  Algorithms, McGraw-Hill, 2007

Course Topics

- An illustrative problem: stable matching
- Basics of algorithm analysis:

  data structures, asymptotic analysis
- Basic graph algorithms
- Greedy algorithms
- Divide and Conquer
- Dynamic Programming
- Network flow
- NP completeness

A First Problem: Stable Matching
[Gale and Shapley, 1962]

- **Motivation.** Design a self-enforcing hiring process:

  For every employer E, and
  for every applicant A who is not scheduled to work for E, either
  1. E prefers all of its accepted applicants to A or
  2. A prefers his/her present situation to working for employer E

  (If the process did not self-enforce these conditions, chaos could result)

A First Problem: Stable Matching

[Gale and Shapley, 1962]

- **Motivation.** Design a self-enforcing hiring process:

  For every employer E, and
  for every applicant A who is not scheduled to work for E, either

  1. E prefers all of its accepted applicants to A or
  2. A prefers his/her present situation to working for employer E

  (If the process did not self-enforce these conditions, chaos could result)

- **Simplifying assumptions:**
  - n employers, n applicants
  - every applicant applies to every employer
  - each employer accepts a single applicant

A First Problem: Stable Matching

[Gale and Shapley, 1962]

- **Motivation.** Design a self-enforcing hiring process:

  For every employer E, and
  for every applicant A who is not scheduled to work for E, either
  1. E prefers all of its accepted applicants to A or
  2. A prefers his/her present situation to working for employer E

  (If the process did not self-enforce these conditions, chaos could result)

- **Simplifying assumptions:**
  - n employers, n applicants
  - every applicant applies to every employer
  - each employer accepts a single applicant

- **Similar applications:**

  assigning medical residents to hospitals, college admissions

- The problem is traditionally described in the context of n men and n women and is sometimes called the stable marriage problem

Stable Matching Problem: Formulation

Given: A set $M = \{m_1, \ldots, m_n\}$ of n men
and a set $W = \{w_1, \ldots, w_n\}$ of n women

- each man $m \in M$ ranks all the women in W
- we say m prefers w to w' if m ranks w higher than w'
- for each m, this ranking of the women is m's preference list
- similarly, each woman $w \in W$ ranks all the men in M

Goal: Find a stable perfect matching

Stable Matching Problem: Formulation

Given: A set $M = \{m_1, \ldots, m_n\}$ of n men
and a set $W = \{w_1, \ldots, w_n\}$ of n women

- each man $m \in M$ ranks all the women in W
- we say m prefers w to w' if m ranks w higher than w'
- for each m, this ranking of the women is m's preference list
- similarly, each woman $w \in W$ ranks all the men in M

Goal: Find a stable perfect matching:

- A matching is a subset S of $M \times W$ such that
  each $m \in M$ and each $w \in W$ appear in at most one pair in S

Stable Matching Problem: Formulation

Given: A set $M = \{m_1, \ldots, m_n\}$ of n men
and a set $W = \{w_1, \ldots, w_n\}$ of n women

- each man $m \in M$ ranks all the women in W
- we say m prefers w to w' if m ranks w higher than w'
- for each m, this ranking of the women is m's preference list
- similarly, each woman $w \in W$ ranks all the men in M

Goal: Find a stable perfect matching:

- A matching is a subset S of $M \times W$ such that
  each $m \in M$ and each $w \in W$ appear in at most one pair in S
- A perfect matching is a matching S such that
  each $m \in M$ and each $w \in W$ appear in *exactly* one pair in S

Stable Matching Problem: Formulation

Given: A set $M = \{m_1, \ldots, m_n\}$ of n men
and a set $W = \{w_1, \ldots, w_n\}$ of n women

- each man $m \in M$ ranks all the women in W
- we say m prefers w to w' if m ranks w higher than w'
- for each m, this ranking of the women is m's preference list
- similarly, each woman $w \in W$ ranks all the men in M

Goal: Find a stable perfect matching:

- A matching is a subset S of $M \times W$ such that
  each $m \in M$ and each $w \in W$ appear in at most one pair in S
- A perfect matching is a matching S such that
  each $m \in M$ and each $w \in W$ appear in *exactly* one pair in S
- an instability in a matching S is a pair (m, w) **not** in S,
  but each of m, w prefers the other to their partner in S

Stable Matching Problem: Formulation

Given: A set $M = \{m_1, \ldots, m_n\}$ of n men
and a set $W = \{w_1, \ldots, w_n\}$ of n women

- each man $m \in M$ ranks all the women in W
- we say m prefers w to w' if m ranks w higher than w'
- for each m, this ranking of the women is m's preference list
- similarly, each woman $w \in W$ ranks all the men in M

Goal: Find a stable perfect matching:

- A matching is a subset S of $M \times W$ such that each $m \in M$ and each $w \in W$ appear in at most one pair in S
- A perfect matching is a matching S such that each $m \in M$ and each $w \in W$ appear in *exactly* one pair in S
- an instability in a matching S is a pair (m, w) **not** in S, but each of m, w prefers the other to their partner in S
- A matching is stable if it contains no instabilities

Stable Marriage Problem: Simple Examples

- Example 1: $n = 2$. Preferences are:

Element	Preference List
m_1	w_1, w_2
m_2	w_1, w_2
w_1	m_1, m_2
w_2	m_1, m_2

- Example 2: $n = 2$. Preferences are:

Element	Preference List
m_1	w_1, w_2
m_2	w_2, w_1
w_1	m_2, m_1
w_2	m_1, m_2

- There may be more than one stable matching!

Stable Matching Problem: Idea behind Algorithm

- Initially, everyone is unmarried
- Suppose an unmarried man m chooses the woman w who is ranked highest on his preference list and proposes to her

  What should w do? Accept? Reject?

Stable Matching Problem: Idea behind Algorithm

- Initially, everyone is unmarried
- Suppose an unmarried man m chooses the woman w who is ranked highest on his preference list and proposes to her

  What should w do? Accept? Reject?

  - she cannot accept the proposal immediately, because another man whom she prefers to m may propose to her later

Stable Matching Problem: Idea behind Algorithm

- Initially, everyone is unmarried

- Suppose an unmarried man m chooses the woman w who is ranked highest on his preference list and proposes to her

  What should w do? Accept? Reject?

  - she cannot accept the proposal immediately, because another man whom she prefers to m may propose to her later

  - she cannot reject the proposal immediately, because she may not receive a proposal from someone she ranks as highly as m

Stable Matching Problem: Idea behind Algorithm

- Initially, everyone is unmarried
- Suppose an unmarried man m chooses the woman w who is ranked highest on his preference list and proposes to her

  What should w do? Accept? Reject?

  - she cannot accept the proposal immediately, because another man whom she prefers to m may propose to her later
  - she cannot reject the proposal immediately, because she may not receive a proposal from someone she ranks as highly as m
  - so they become engaged (the engagement may be broken later)

Stable Matching Problem:
Sketch of the Gale-Shapley Algorithm

- Initially, everyone is free (*i.e.*, not engaged)

Stable Matching Problem: Sketch of the Gale-Shapley Algorithm

- Initially, everyone is free (*i.e.*, not engaged)

- The following step is performed repeatedly
  - some free man m chooses the highest-ranked woman w (on his preference list) to whom he has not yet proposed, and proposes to her
  - if w is also free, then m and w become engaged

Stable Matching Problem: Sketch of the Gale-Shapley Algorithm

- Initially, everyone is free (*i.e.*, not engaged)

- The following step is performed repeatedly
  - some free man m chooses the highest-ranked woman w (on his preference list) to whom he has not yet proposed, and proposes to her
  - if w is also free, then m and w become engaged
  - if w is engaged to some other man m', then she determines whether m or m' ranks higher on her preference list
  - she becomes engaged to whoever ranks higher, and the other man becomes free

Stable Matching Problem: Sketch of the Gale-Shapley Algorithm

- Initially, everyone is free (*i.e.*, not engaged)

- The following step is performed repeatedly
  - some free man m chooses the highest-ranked woman w (on his preference list) to whom he has not yet proposed, and proposes to her
  - if w is also free, then m and w become engaged
  - if w is engaged to some other man m', then she determines whether m or m' ranks higher on her preference list
  - she becomes engaged to whoever ranks higher, and the other man becomes free

- When there are no free men, the algorithm terminates and all engagements are declared final

Stable Matching Problem:
Pseudocode for the Gale-Shapley Algorithm

```
while there is a man who is free and has not proposed to every woman
   m ← such a man
   w ← the highest ranking women in m's preference list
            to whom m has not yet proposed




endwhile
return the set S of engaged pairs
```

Stable Matching Problem: Pseudocode for the Gale-Shapley Algorithm

```
while there is a man who is free and has not proposed to every woman
   m ← such a man
   w ← the highest ranking women in m's preference list
          to whom m has not yet proposed
   if w is free then (m, w) become engaged
   else // w is currently engaged to some other man, m'
      if w prefers m' to m then m remains free
      else // w prefers m to m'
         (m, w) become engaged
         m' becomes free
      endif
   endif
endwhile
return the set S of engaged pairs
```

Stable Matching Problem: Analysis of G-S Algorithm

- **Correctness:**
  Does the algorithm compute a stable matching?

- **Efficiency:**
  How efficient is the algorithm?

Stable Matching Problem: Analysis of G-S Algorithm

- A woman w
  - initially, not engaged
  - once she receives a proposal, she always remains engaged
  - the sequence of partners to whom she is engaged gets better

    (with respect to her preference list)

Stable Matching Problem: Analysis of G-S Algorithm

- A woman w
  - initially, not engaged
  - once she receives a proposal, she always remains engaged
  - the sequence of partners to whom she is engaged gets better
    (with respect to her preference list)

- A man m
  - initially, not engaged
  - alternates between being free and engaged
  - proposes to each woman at most once
  - the sequence of partners to whom he proposes gets worse
    (with respect to his preference list)

Stable Matching Problem: Analysis of G-S Algorithm

- A woman w
  - initially, not engaged
  - once she receives a proposal, she always remains engaged
  - the sequence of partners to whom she is engaged gets better
    (with respect to her preference list)

- A man m
  - initially, not engaged
  - alternates between being free and engaged
  - proposes to each woman at most once
  - the sequence of partners to whom he proposes gets worse
    (with respect to his preference list)

Claim: The G-S algorithm terminates after at most n^2 iterations.
Proof: count proposals

Stable Matching Problem:
Analysis of G-S Algorithm (continued)

Claim: If a man m is free,
there is some woman to whom he has not yet proposed.

Proof: by contradiction

Stable Matching Problem:
Analysis of G-S Algorithm (continued)

Claim: If a man m is free,
there is some woman to whom he has not yet proposed.

Proof: by contradiction

- suppose m is free and has proposed to all women
- then all women are engaged
- but since $|M| = |W|$, this means all men are engaged, a contradiction

Stable Matching Problem: Analysis of G-S Algorithm (continued)

Claim: If a man m is free,
there is some woman to whom he has not yet proposed.

Proof: by contradiction

- suppose m is free and has proposed to all women
- then all women are engaged
- but since $|M| = |W|$, this means all men are engaged, a contradiction

Claim: The matching computed by the G-S algorithm
is a perfect matching.

Proof by contradiction

Stable Matching Problem: Analysis of G-S Algorithm (continued)

Claim: If a man m is free,
there is some woman to whom he has not yet proposed.

Proof: by contradiction

- suppose m is free and has proposed to all women
- then all women are engaged
- but since $|M| = |W|$, this means all men are engaged, a contradiction

Claim: The matching computed by the G-S algorithm
is a perfect matching.

Proof by contradiction

- suppose there is a free man m when the algorithm terminates
- by the code, m must have proposed to every woman
- by the previous fact, this is impossible

Stable Matching Problem: Analysis of G-S Algorithm (continued)

Claim: The matching computed by the G-S algorithm is a stable matching.

Proof by contradiction

Stable Matching Problem: Analysis of G-S Algorithm (continued)

Claim: The matching computed by the G-S algorithm is a stable matching.

Proof by contradiction

- Suppose that there is an instability (m, w')
- This means that the matching contains two pairs (m, w) and (m', w') such that m prefers w' to w, and w' prefers m to m'

Stable Matching Problem: Analysis of G-S Algorithm (continued)

Claim: The matching computed by the G-S algorithm is a stable matching.

Proof by contradiction

- Suppose that there is an instability (m, w')
- This means that the matching contains two pairs (m, w) and (m', w') such that m prefers w' to w, and w' prefers m to m'
- The last time m proposed, it was to w.
  Did m propose to w' at some earlier time?
- If No: then m does not prefer w' to w, a contradiction

Stable Matching Problem: Analysis of G-S Algorithm (continued)

Claim: The matching computed by the G-S algorithm is a stable matching.

Proof by contradiction

- Suppose that there is an instability (m, w')
- This means that the matching contains two pairs (m, w) and (m', w') such that m prefers w' to w, and w' prefers m to m'
- The last time m proposed, it was to w. Did m propose to w' at some earlier time?
- If No: then m does not prefer w' to w, a contradiction
- If Yes: w' rejected m for some other man m''. Since every woman's engagement partners improve, this means that w' prefers her final partner (*i.e.*, m') to m, a contradiction.

Stable Matching Problem: Determinacy in G-S Algorithm

- The G-S algorithm is non-deterministic
  Why? What does that mean?

Stable Matching Problem:
Determinacy in G-S Algorithm

- The G-S algorithm is non-deterministic
- Nevertheless, it always computes the same stable matching on any particular input (deterministic result)
  - w is a valid partner of m if (m, w) is in some stable matching

Stable Matching Problem: Determinacy in G-S Algorithm

- The G-S algorithm is non-deterministic
- Nevertheless, it always computes the same stable matching on any particular input (deterministic result)
  - w is a valid partner of m if (m, w) is in some stable matching
  - w is the best valid partner of m if
    1) w is a valid partner of m, and
    2) no woman who m ranks higher than w is a valid partner of m

Stable Matching Problem: Determinacy in G-S Algorithm

- The G-S algorithm is non-deterministic
- Nevertheless, it always computes the same stable matching on any particular input (deterministic result)
  - w is a valid partner of m if (m, w) is in some stable matching
  - w is the best valid partner of m if
    1) w is a valid partner of m, and
    2) no woman who m ranks higher than w is a valid partner of m
  - the G-S algorithm always pairs each man with his *best* valid partner (proof: see text)
  - the G-S algorithm always pairs each woman with her *worst* valid partner (proof: see text)

Analysis of G-S Algorithm

- What we have not yet discussed:
  - how much time does the G-S algorithm require?
  - how does the time required by the G-S algorithm scale up with the size of the problem?

Analysis of G-S Algorithm

- What we have not yet discussed:
  - how much time does the G-S algorithm require?
  - how does the time required by the G-S algorithm scale up with the size of the problem?

- We will come back to this

- First, we need a computational model and a metric for measuring the time requirement of algorithms

Random Access Machine (RAM)

- Primitive operations on "words" include:
  - assigning a value to a variable
  - performing an arithmetic or Boolean operation

Random Access Machine (RAM)

- Primitive operations on "words" include:
  - assigning a value to a variable
  - performing an arithmetic or Boolean operation
  - comparing two numbers
  - indexing into an array

Random Access Machine (RAM)

- Primitive operations on "words" include:
  - assigning a value to a variable
  - performing an arithmetic or Boolean operation
  - comparing two numbers
  - indexing into an array
  - following an object reference / dereferencing a pointer
  - calling or returning from a function/method
  - branching

Random Access Machine (RAM)

- Primitive operations on “words” include:
  - assigning a value to a variable
  - performing an arithmetic or Boolean operation
  - comparing two numbers
  - indexing into an array
  - following an object reference / dereferencing a pointer
  - calling or returning from a function/method
  - branching

- To measure running time: count operations
  this is sometimes called Unit-Cost RAM

Example: Sequential Search in an Array

Input: An array $A[n]$, where $n \geq 1$; an item x
Output: Index where x occurs in A, or -1

```
for i ← 0 to n-1 do
   if A[i] = x then return(i)
return(-1)
```

which is actually implemented as

```
  i ← 0                          1
label:
  if A[i] = x then return(i)     2 or 3
  i ← i + 1                      2
  if (i < n) goto label          2 or 1
  return(-1)                     1
```

Time is between 4 and $6n + 1$ units. Too much detail?

Sequential Search, continued

In the previous example, all we really need to know for most practical purposes is:

> *The running time of sequential search on an input of size n is, in the worst case, proportional to n.*

Sequential Search, continued

In the previous example, all we really need to know for most practical purposes is:

The running time of sequential search on an input of size n is, in the worst case, proportional to n.

A shorthand way of saying this is:

Sequential search runs in $O(n)$ time.

We will give a formal definition of this "big oh" notation later.

Similarly, we can talk about algorithms that run in

- $O(n^2)$ time
- $O(\log\ n)$ time
- $O(1)$ time (constant time)
- ...

Some issues explored in the homework problems

- What if there are more men than women, or if marriages are not 1-1. For example, assigning residents to hospitals. How should the algorithm be modified to handle this case?

Some issues explored in the homework problems

- What if there are more men than women, or if marriages are not 1-1. For example, assigning residents to hospitals. How should the algorithm be modified to handle this case?

- What about indifference (ties) in the rankings? How to define stability? Is there still always a stable matching?

Some issues explored in the homework problems

- What if there are more men than women, or if marriages are not 1-1. For example, assigning residents to hospitals. How should the algorithm be modified to handle this case?

- What about indifference (ties) in the rankings? How to define stability? Is there still always a stable matching?

- Is there ever a situation where a man (or woman) could benefit by pretending that his/her ranking is different than it actually is?

Some issues explored in the homework problems

- What if there are more men than women, or if marriages are not 1-1. For example, assigning residents to hospitals. How should the algorithm be modified to handle this case?

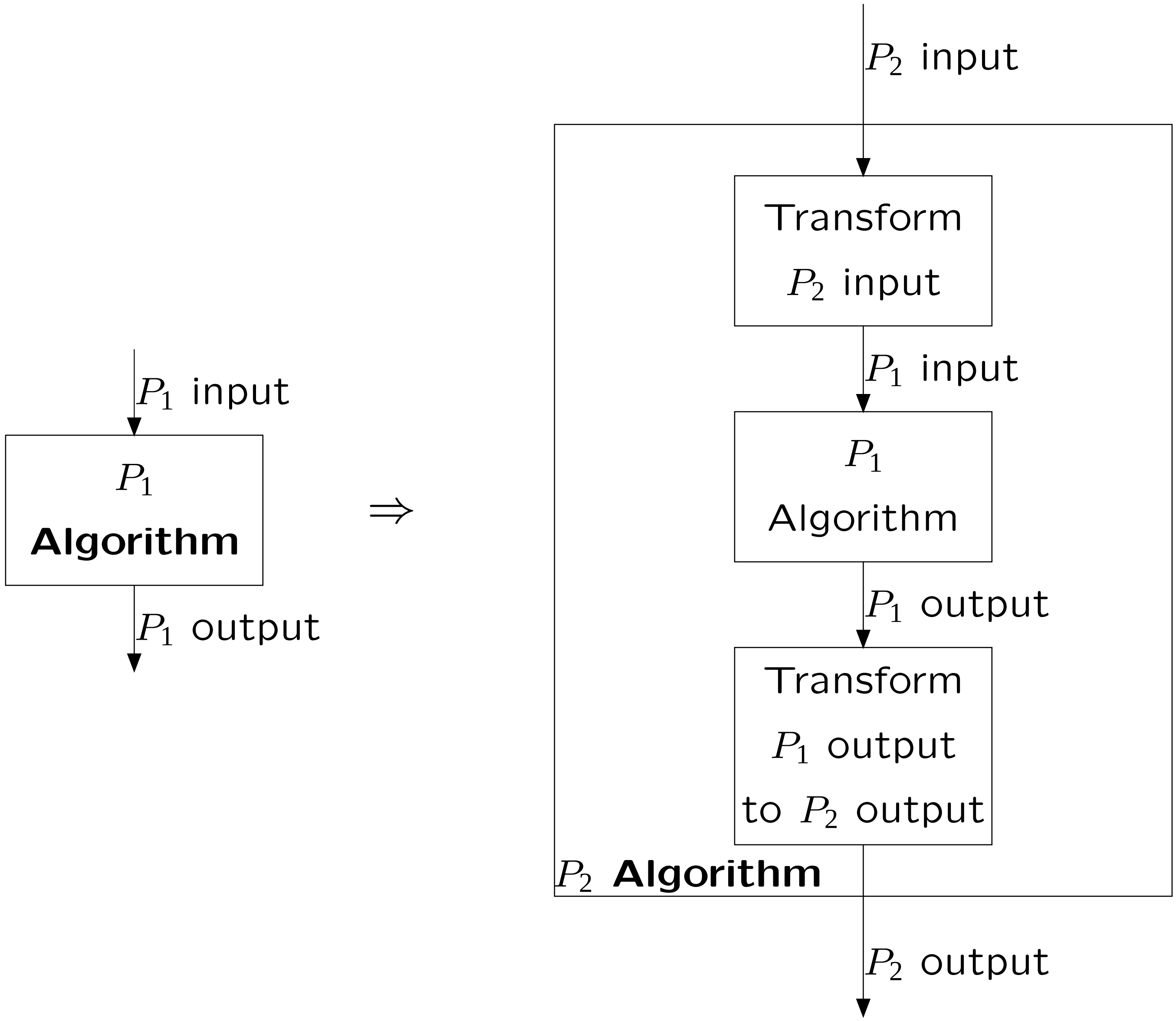
- What about indifference (ties) in the rankings? How to define stability? Is there still always a stable matching?

- Is there ever a situation where a man (or woman) could benefit by pretending that his/her ranking is different than it actually is?

- Applications of stable matching

Problem reduction: Reduce P_2 to P_1

Suppose we have an algorithm for problem P_1 and want to adapt it to solve problem P_2.



Problem reduction: Reduce P_2 to P_1

- Practical advantages of reducing P_2 to P_1 (rather than modifying the source code of P_1 to obtain new source code for P_2):
  - "Software/component reuse"
  - "Modular design"
  - "Don't reinvent the wheel"

Problem reduction: Reduce P_2 to P_1

- Practical advantages of reducing P_2 to P_1 (rather than modifying the source code of P_1 to obtain new source code for P_2):
  - "Software/component reuse"
  - "Modular design"
  - "Don't reinvent the wheel"

- Reduction is useful for analysis of algorithms

  If the two I/O transformation boxes are "efficient" then we can conclude:

  1. $\exists$ an efficient algorithm for $P_1 \Rightarrow \exists$ an efficient algorithm for P_2
  2. $\nexists$ an efficient algorithm for $P_2 \Rightarrow \nexists$ an efficient algorithm for P_1