CPSC 320: Steps in Algorithm Design and Analysis*

In this worksheet, you'll practice five useful steps for designing and analyzing algorithms, starting from a possibly vague problem statement. These steps will be useful throughout the class. They could also be useful when you find yourself thinking on your feet in an interview situation. And hopefully they will serve you well in your work post-graduation too!

We'll use the **Stable Matching Problem (SMP)** as our working example. Following the historical literature, the text formulates the problem in terms of marriages between men and women. We'll avoid the gender binaries inherent in that literature and use employers and job applicants instead. Imagine for example the task faced by UBC's co-op office each semester, which seeks to match hundreds of student applicants to employer internships. To keep the problem as simple as possible for now, assume that every applicant has a full ranking of employers and vice versa (no ties).

Step 1: Build intuition through examples.

1. **Write down small and trivial instances of the problem.**
   We'll use the words "instance" and "inputs" interchangeably.

Slack : Fiona Emil
Etsy : Emil Fiona

Fiona : Slack Etsy
Emil : Slack Etsy

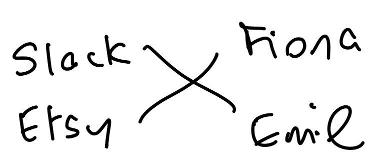
e_1 : a_2 a_1 a_3
e_2 : a_1 a_2 a_3
e_3 : a_2 a_1 a_3

a_1 : e_2 e_1 e_3
a_2 : e_3 e_2 e_1
a_3 : e_2 e_1 e_3

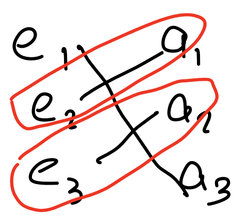
Trivial instance: 1 empl., 1 appl.
0 " 0 "

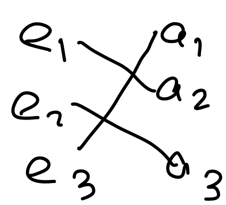
2. **Write down potential solutions for your instances.**
   Are some solutions better than others? How so?

Slack — Fiona
Etsy — Emil



e_1 – a_1
e_2 – a_2
e_3 – a_3





Step 2: Develop a formal problem specification

1. **Develop notation for describing a problem instance.** What quantities or data (such as numbers, sets, lists, etc.) matter? Give them short, usable names. Think of these as input parameters to the algorithm code. Use your earlier examples to illustrate your notation.

n : # applicants = # employers

$E = \{e_1, e_2, \ldots, e_n\}$ set of employers

$A = \{a_1, a_2, \ldots, a_n\}$ " " applicants

$P_E[i]$: empoyer e_i's ranked list of applicants.

P_E : list of lists (2D array)

$P_A[j]$: applicant a_j's ranked list of employers

P_A ...

$P_A[j][k]$ is the employe of rank k on applicant a_j's list

2. **Develop notation for describing a potential solution.**
Use your earlier examples to illustrate your notation.

A potential solution is a set M of pairs of form $(e, a) \in E \times A$

The set has size n

Every employer and applicant is in exactly one pair.

Such a set M is called a pefect matching.

3. **Describe what you think makes a solution *good*.**

Maximize #applicants and employers that get their first choice.

Minimize ... ... their last choice.

3. **Describe what you think makes a solution *good*.**

Idea suggested last time:

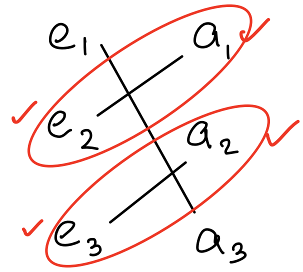
Maximize # applicants and employers that get their <u>first</u> choice.

<u>Example from last class:</u>

$e_1 : a_2\ a_1\ a_3$ $\qquad$ $a_1 : e_2\ e_1\ e_3$

$e_2 : a_1\ a_2\ a_3$ $\qquad$ $a_2 : e_3\ e_2\ e_1$

$e_3 : a_2\ a_1\ a_3$ $\qquad$ $a_3 : e_2\ e_1\ e_3$



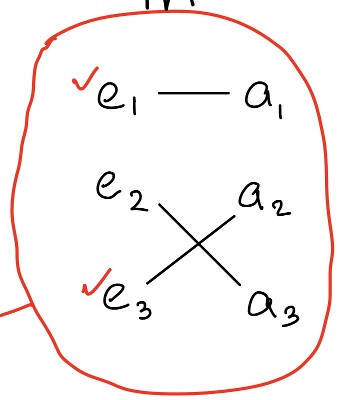
<u>Another example:</u>

$e_1 : a_1\ a_2\ a_3$ $\qquad$ $a_1 : e_3\ e_2\ e_1$

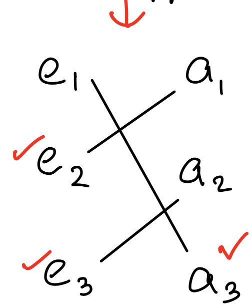
$e_2 : a_1\ a_2\ a_3$ $\qquad$ $a_2 : e_1\ e_2\ e_3$

$e_3 : a_2\ a_1\ a_3$ $\qquad$ $a_3 : e_1\ e_2\ e_3$

M



M'



Consider e_2 and a_1

e_2 prefers a_1 to its match a_3

a_1 " e_2 " " " e_1

We call this an <u>instability</u>.

More generally pair $(e, a) \in E \times A$ is an instability

if $(e, a) \notin M$, a' is matched with e

e' " " " a

but e prefers a to a'

and a " e to e'

Gale-Shapley criterion for <u>good</u> solution: no instabilities.

Clicker question #0

Are you able to download iClicker Cloud, go to our class, and answer this question?

A. Yes

B. No

C. Bacon

(Most of you should already have been added through Canvas. If not, search for “9 AM CPSC 320” or “3 PM CPSC 320.”)

Clicker question #1

Which employer/applicant pair forms an instability in the matching below?

$$M$$

$$\begin{array}{lcccl}
e_1: a_1, a_3, a_2 & : & e_1 \;\text{———}\; a_1 & : & e_1, e_3, e_2 \\
e_2: a_1, a_2, a_3 & : & e_2 \;\text{———}\; a_2 & : & e_1, e_3, e_2 \\
e_3: a_1, a_2, a_3 & : & e_3 \;\text{———}\; a_3 & : & e_1, e_2, e_3 \\
e \quad\quad a \quad a' & & & &
\end{array}$$

A. (e_3, a_1)

B. (e_3, a_2) $\notin M$

C. (e_3, a_3) $\in M$

D. None of these are unstable

$e = e_3 \quad a = a_2 \quad (e, a) \notin M.$

e is matched with $a_3 = a'$ $\quad$ e prefers a to a'

a is " " $e_2 = e'$

Step 3: Identify similar problems. What are the similarities?

... maybe bipartite matching

CaRMP (Canadian residents matching problem) (see Piazza post). It's more general than our problem, e.g. an employer (hospital) may want more than one applicant.

Step 4: Evaluate simple algorithmic approaches, such as brute force.

1. **Design a brute force algorithm for the SMP problem.**
Given as input a problem instance, a "brute force" algorithm enumerates all potential solutions, and checks each to see if it is good. If a good solution is found, the algorithm outputs this solution, and otherwise reports that there is no good solution. **Flesh out the details: what is a problem instance, and potential solution in this case? How would you test if a potential solution is a good solution?**

procedure SMP-Brute-Force (n, P_E, P_A)

    for each perfect matching M   ▷ potential solution
        if M has no instabilities   ▷ it is a good solution
            return M
    return "no good solution"

$O(n^2)$

$O(n^2)$ procedure IsStable (M)   ▷ test if a pot. soln. is a good soln

    for each e ∈ E   } n²
        for each a ∈ A   ▷ test if (e,a) is instability
            if (e,a) ∉ M   } O(1)
                find a' such that (e,a') ∈ M
                 "   e'   "   " (e',a) ∈ M
                if (e prefers a to a') and (a prefers e to e')
                    return False
    return True

2. **Analyze the worst case running time of brute force.**

   - **Bound the running time, say $t(n)$, of your procedure to test if a potential solution is a good solution. Assume that the input M is represented as a list of pairs (e, a) such that e is matched with a, where $1 \le e, a \le n$.**

n times — for each $e \in E$

n times — for each $a \in A$ ▷ test if (e,a) is instability

if $(e,a) \notin M$

find a' such that $(e,a') \in M$

" e' " " $(e',a) \in M$

if (e prefers a to a') and (a prefers e to e')

return False

return True

$\boxed{\text{if } (e,a) \notin M}$

Assume that $e \in \{1, 2, \ldots, n\}$

$a \in \{1, 2, \ldots, n\}$

Represent M as an array to get $O(1)$ runtime:

$Match_E[1 \ldots n]$, where $Match_E[e] = a$ iff $(e,a) \in M$

if $Match_E[e] \neq a$

[List implementation could be $\Theta(n)$ time in worst case:

M represented as list: $(e_1, a_5), (e_2, a_7) \ldots$

example: $(1, 5), (2, 7) \ldots (n, 3)$]

for each $e \in E$
    for each $a \in A$ ▷ test if (e,a) is instability
        if $(e,a) \notin M$
            ① find a' such that $(e,a') \in M$
            ② " e' " " $(e',a) \in M$
            if (e prefers a to a') and (a prefers e to e')
                return False
return True

① Let $a' \leftarrow \text{Match}_E[e]$

② New array Match_A

Let $e' \leftarrow \text{Match}_A[a]$

[Using ranking lists takes $O(n)$ time.]

Array approach: 2D array

$\text{Rank}_A[1\ldots n][1\ldots n]$. $\text{Rank}_A[a][e]$ is the rank of e in a's list.

$\text{Rank}_A[a][e] = 3$ if e is third on a's list.

" a prefers e to e' " "$\text{Rank}_A[a][e] < \text{Rank}_A[a][e']$"

??

- **How many potential solutions are there?** Each potential solution is a perfect matching. Imagine that the n applicants are arranged in some order. How many different ways can we arrange (permute) the n employers next to them?

applicants: 1 2 3 .... n

matches: (employers) 1 2 ⋮ n 2 ⋮ n

#choices: $n \quad (n-1) \quad (n-2) \quad \cdots \quad 1 = n!$

Clicker question #2

- **Putting the previous two parts together, what can you say about the overall worst-case running time of brute force? Assume that the time needed to generate each potential solution is less than that needed to check if a potential solution is a good solution.**

Big-O notn, short answer.

$O(\underbrace{n! \cdot n^2}_{\text{\#times iterate for loop}})$

[is this just $O(n!)$? No :C]

Step 5: Design a better algorithm.

1. **Brainstorm some ideas, then sketch out an algorithm.**
   Carefully try out your algorithm on your examples, and design instances that challenge the correctness of your approach.

Gale-Shapley algorithm from handout:

```
set all a ∈ A and e ∈ E to free
while some free employer e hasn't made an offer to every student do
    a ← the highest-ranking student e hasn't made an offer to
    if a is free then
        hire(e, a)
    else
        e′ ← a's current employer
        if a prefers e to e′ then
            set e′ to free
            hire(e, a)
        endif
    endif
endwhile
return the set of pairs
```

Example:

$e_1 : a_1 \ a_2 \ a$	$a_1 : e_3 \ e_2 \ e_1$
$e_2 : a_1 \ a_2 \ a_3$	$a_2 : e_1 \ e_2 \ e_3$
$e_3 : a_2 \ a_1 \ a_3$	$a_3 : e_1 \ e_2 \ e_3$

Clicker question #3

In the SMP instance:

$$\begin{array}{ccc} a_1, a_2 & : & e_1 \\ a_2, a_1 & : & e_2 \end{array} \qquad \begin{array}{ccc} a_1 & : & e_2, e_1 \\ a_2 & : & e_1, e_2 \end{array}$$

Which a should be paired with e_1?

A. a_1
B. a_2