Éléments d'algorithmique

Mariages stables

François Pottier

4 juin 2013

Brève présentation

Je suis chercheur à l'INRIA, spécialiste de la théorie des langages de programmation.

Je suis également (à temps partiel) professeur chargé de cours à l'École Polytechnique, où j'enseigne « Algorithmique et programmation » en deuxième année (L3), en collaboration avec Benjamin Werner.

Que dit le programme ?

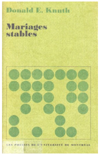
Dans le programme des classes préparatoires, j'ai relevé quelques phrases-clef :

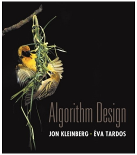
- analyser un problème ;
- concevoir un algorithme répondant à un problème précisément posé ;
- justifier qu'un algorithme termine et produit l'effet attendu ;
- prédire l'efficacité d'un algorithme ;
- traduire un algorithme dans un langage de programmation, et à cette fin, choisir des structures de données appropriées.

Comment aborder tout cela ?

Je souhaite aborder ces thèmes à travers un exemple.

Knuth (1976) a noté l'intérêt du problème des mariages stables en tant qu'introduction à l'algorithmique.





Kleinberg et Tardos (2005) en font le chapitre introductif de leur livre.

Quels pré-requis ?

En termes mathématiques, ensembles et relations suffisent pour énoncer et étudier le problème.

En termes de programmation, cet exemple mobilise à peu près toutes les notions du programme : variables, tableaux, conditionnelles, boucles.

En termes d'algorithmique, il permet d'aborder tous les thèmes cités plus tôt : analyser le problème, concevoir et justifier un algorithme, le traduire vers un langage de programmation, prédire son efficacité.

Quelle difficulté ?

Pour des élèves d'« informatique pour tous », c'est probablement un algorithme difficile, que l'on pourrait présenter en deuxième année.

Ici, il illustre les questions auxquelles l'algorithmique tente de répondre.

Deux ensembles d'individus

Soient $\mathcal{H}$ et $\mathcal{F}$ deux ensembles de même cardinal n.

On les appelle de façon imagée « hommes » et « femmes ».

On pourrait aussi parler de « patients » et « d'hôpitaux », ou encore « d'étudiants » et « d'universités ».

Couplages

Le but du jeu va être d'associer hommes et femmes.

Chaque homme peut être marié à au plus une femme, et vice-versa.

Définition

Un couplage c est une relation fonctionnelle et injective entre $\mathcal{H}$ et $\mathcal{F}$.

En d'autres termes, c'est une bijection entre un sous-ensemble de $\mathcal{H}$ et un sous-ensemble de $\mathcal{F}$.

Couplages parfaits

On souhaite que personne ne reste célibataire.

Définition

Un couplage est parfait si c'est une bijection entre $\mathcal{H}$ et $\mathcal{F}$.

Chacun exprime des préférences

On suppose que chaque homme classe toutes les femmes par ordre de préférence, et inversement.

Pour chaque $h \in \mathcal{H}$, on a donc un ordre total $\leq_h$ sur $\mathcal{F}$, et pour chaque $f \in \mathcal{F}$, un ordre total $\leq_f$ sur $\mathcal{H}$.

On note $f_1 \leq_h f_2$ si h préfère f_1 à f_2.

À quoi servent ces préférences ?

L'idée n'est pas d'attribuer à chacun son partenaire préféré : c'est bien sûr impossible en général.

L'idée est de tenir compte des préférences pour proposer un couplage parfait stable.

Si l'ordinateur central a décidé quel étudiant ira dans quelle école, on veut éviter que des négociations en sous-main puissent remettre en cause ces décisions.

Instabilité

Un couplage c présente une instabilité si un homme h_1 et une femme f_2 sont tous deux prêts à quitter leur partenaire actuel pour s'associer.

Définition

Si h_1 — $f_1 \in c$ et h_2 — $f_2 \in c$ et $f_2 <_{h_1} f_1$ et $h_1 <_{f_2} h_2$, alors la paire h_1 — f_2 est une instabilité pour c, et c est instable.

Stabilité

A contrario, la stabilité peut s'écrire ainsi :

$$h_1 \text{—} f_1 \in c \wedge h_2 \text{—} f_2 \in c \wedge f_2 <_{h_1} f_1 \Rightarrow h_2 <_{f_2} h_1$$

On peut également l'écrire avec deux inégalités au sens large.

À chaque fois que h_1 aimerait quitter sa femme pour f_2, malheureusement pour lui f_2 préfère ne pas quitter son mari.

Énoncé du problème

Étant données les préférences (arbitraires) des uns et des autres,

- existe-t-il toujours un couplage parfait stable ou c.p.s. ?
- si oui, comment en trouver un (efficacement si possible) ?

On peut aussi se demander si ce couplage, lorsqu'il existe, est unique.

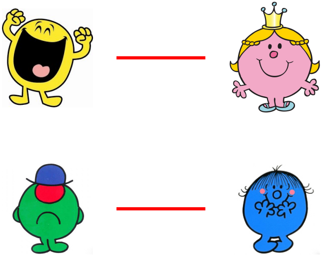
Quelques exemples

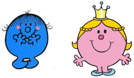
L'étude de quelques cas particuliers peut aider à former une intuition.

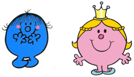
Exemple 1 : où les hommes s'accordent

Si tous les hommes ont les mêmes préférences, alors la femme la plus prisée est libre de son choix.

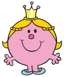
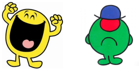
Il existe un unique couplage parfait stable.



 préfère : 

 préfère : 

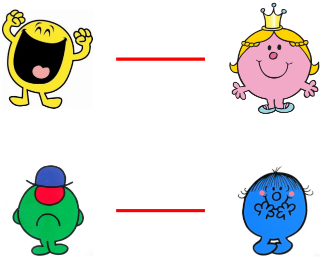
 préfère : 

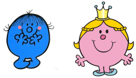
 préfère : 

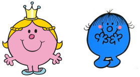
Exemple 2 : où l'attraction est mutuelle

Si *h* préfère *f* exactement lorsque *f* préfère *h*, chacun et chacune est libre de choisir son favori.

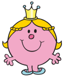
Dans ce cas aussi, il existe un unique couplage parfait stable.



 préfère : 

 préfère : 

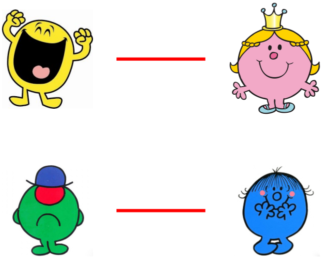
 préfère : 

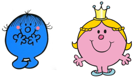
 préfère : 

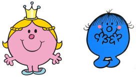
Exemple 3 : où l'amour n'est pas réciproque

Ici, il existe deux couplages parfaits, et tous deux sont stables.

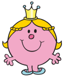
L'un satisfait les hommes, l'autre les femmes.



 préfère : 

 préfère : 

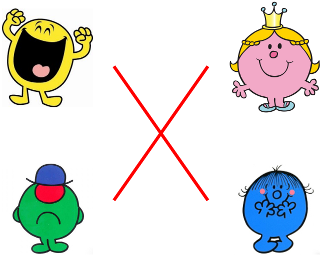
 préfère : 

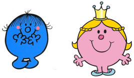
 préfère : 

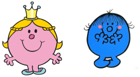
Exemple 3 : où l'amour n'est pas réciproque

Ici, il existe deux couplages parfaits, et tous deux sont stables.

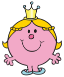
L'un satisfait les hommes, l'autre les femmes.



 préfère : 

 préfère : 

 préfère : 

 préfère : 

Un cas particulier

Supposons que tous les hommes aient les mêmes préférences.

On a donc un ordre total $\leq$ sur $\mathcal{F}$, qui ne dépend plus de h.

Dans ce cas la stabilité s'écrit :

$$h_1 — f_1 \in c \wedge h_2 — f_2 \in c \wedge f_2 \leq f_1 \Rightarrow h_2 \leq_{f_2} h_1$$

Autrement dit, « une femme plus désirée impose son choix ».

On voit alors (et les élèves voient, j'espère) qu'un certain couple va nécessairement se former...

Une hypothèse simplificatrice

Dans un c.p.s., la femme f la plus désirée (minimale pour $\leq$) ne peut être associée qu'à l'homme qu'elle préfère (minimal pour $\leq_f$).

Inversement, dans un couplage quelconque, si elle est associée à cet homme, alors elle ne participe à aucune instabilité.

Nous allons en déduire existence et unicité du couplage parfait stable.

La preuve est par récurrence et peut s'exprimer comme un algorithme...

Un cas particulier

On peut écrire cet algorithme sous forme récursive :

fonction $\textsc{UniqueCouplage}(\mathcal{H}, \mathcal{F})$
  soit n le cardinal des ensembles $\mathcal{H}$ et $\mathcal{F}$
  si $n = 0$ **alors**
    renvoyer le couplage vide
  sinon
    soit f la femme la plus désirée parmi $\mathcal{F}$
    soit h l'homme préféré de f parmi $\mathcal{H}$
    soit c le couplage $\textsc{UniqueCouplage}(\mathcal{H} \setminus \{h\}, \mathcal{F} \setminus \{f\})$
    renvoyer $c \cup \{h — f\}$

(Les ordres $\leq$ et $\leq_f$ sont supposés fixés.)

L'algorithme s'exécute-t-il sans erreur ?

Sous l'hypothèse que $\mathcal{H}$ et $\mathcal{F}$ sont de même cardinal, l'appel $\textsc{UniqueCouplage}(\mathcal{H}, \mathcal{F})$ s'exécute sans erreur, car :

- n est bien défini ;
- h et f sont pris parmi un ensemble non vide ;
- l'appel récursif $\textsc{UniqueCouplage}(\mathcal{H} \setminus \{h\}, \mathcal{F} \setminus \{f\})$ satisfait à son tour l'hypothèse que les deux paramètres sont de même cardinal.

Une condition nécessaire à un appel est appelée précondition.

L'algorithme termine-t-il ?

On démontre par récurrence que, quel que soit le cardinal n de $\mathcal{H}$ et $\mathcal{F}$, l'appel UniqueCouplage($\mathcal{H}, \mathcal{F}$) termine :

- si $n = 0$, il termine immédiatement.
- si $n > 0$, l'algorithme calcule f, puis h, puis effectue un appel récursif pour un cardinal $n - 1$, qui par hypothèse de récurrence termine ; alors, il a terminé.

Le cardinal n est un variant : un entier naturel, fonction des paramètres (ici $\mathcal{H}$ et $\mathcal{F}$), qui décroît strictement à chaque appel récursif.

L'algorithme est-il correct ?

On démontre par récurrence que, quel que soit le cardinal n de $\mathcal{H}$ et $\mathcal{F}$, le résultat de UNIQUECOUPLAGE$(\mathcal{H}, \mathcal{F})$ est un c.p.s., et est l'unique c.p.s..

- si $n = 0$, c'est immédiat.
- si $n > 0$, par hypothèse de récurrence, l'appel récursif produit l'unique c.p.s. pour $\mathcal{H} \setminus \{h\}$ et $\mathcal{F} \setminus \{f\}$. Il reste à argumenter que :
  - si on lui ajoute $h — f$, on obtient un c.p.s. pour $\mathcal{H}$ et $\mathcal{F}$;
  - tout c.p.s. pour $\mathcal{H}$ et $\mathcal{F}$ contient $h — f$.

Une garantie à propos d'un résultat est appelée postcondition.

Preuve = programme

Nous avons donc démontré de façon effective :

Théorème

Si tous les hommes ont le même ordre de préférences, alors il existe un unique couplage parfait stable.

Dans un cours de mathématiques, on pourrait démontrer cela directement, par récurrence, sans mentionner le mot « algorithme ».

Dans un cours d'informatique, on peut souligner le fait que la preuve de ce résultat est un programme qui construit le couplage recherché.

Itération ou récursivité ?

On pourrait aussi écrire cet algorithme à l'aide d'une boucle.

Il faudrait faire apparaître une variable modifiable c contenant le couplage en cours de construction.

Au lieu de raisonner en termes de pré- et post-conditions, il faudrait énoncer un invariant de boucle.

Itération ou récursivité ?

Le programme officiel demande que l'on « présente les avantages et inconvénients » de la récursivité.

Quels sont-ils ? Voici une liste personnelle et non exhaustive :

+ Elle est naturelle : preuve par récurrence = programme récursif ;
+ Elle peut permettre de se passer de variables modifiables ;
– Elle peut sembler déroutante si on l'explique de façon trop mécaniste ;
– Il faut savoir que chaque appel consomme un espace $O(1)$.

Itération ou récursivité ?

Le programme parle de « comprendre le fonctionnement d'un algorithme récursif et l'utilisation de la mémoire lors de son exécution ».

Il me semble qu'il est important de :

- d'abord comprendre les algorithmes récursifs ;
- ensuite seulement présenter leur fonctionnement, c'est-à-dire comment la machine les exécute, pas à pas, à l'aide d'une pile.

Ce n'est pas fini...

On pourrait poser d'autres questions encore :

- comment traduire cet algorithme en Java ou Python, par exemple ?
- quelle est alors sa complexité en temps et en espace ?

J'aborderai ces questions plus loin.

Un problème plus difficile

Dans le cas général, on ne voit pas comment exhiber une paire h — f qui appartient certainement à un c.p.s..

On ne sait même pas a priori s'il existe un c.p.s. !

On imagine donc que l'algorithme désiré devra non seulement ajouter des paires à un couplage c en cours de construction, mais aussi en retirer.

Mais cela, suivant quelle stratégie ?

Une stratégie chaotique ?

Si on laissait les couples se recombiner au hasard des instabilités ?

On obtiendrait un processus non déterministe :

> **soit** c un couplage parfait quelconque
> **tant que** c présente une instabilité $h_1 — f_2$ **faire**
>   **soit** f_1 la fiancée de h_1
>   **soit** h_2 le fiancé de f_2
> $c \leftarrow c \setminus \{h_1 — f_1, h_2 — f_2\} \cup \{h_1 — f_2, h_2 — f_1\}$

Ce processus est-il un algorithme ? i.e., termine-t-il toujours ?

Non. (Contre-exemple omis.)

La stratégie de Gale et Shapley

Gale et Shapley ont proposé en 1962 une meilleure approche.

On fait évoluer un couplage partiel (i.e., pas parfait), initialement vide.

On lui ajoute et retire des paires suivant une stratégie qui garantit :

- terminaison ;
- perfection et stabilité du couplage final.

Comment garantir la terminaison ?

L'ajout de chaque paire h — f sera considéré une fois au plus.

Ainsi, il est clair que l'algorithme terminera.

Comment garantir la stabilité ?

On tient compte des préférences, et cela de deux manières.

- l'ordre dans lequel les paires h — f sont considérées est dicté par les préférences ;
- pour préserver la bijectivité, il faut parfois choisir entre une ancienne liaison et une nouvelle ; ce choix est alors dicté par les préférences.

Comment garantir la stabilité ?

Plus précisément, on organise les choses de façon asymétrique :

- l'ordre dans lequel les paires h — f sont considérées est dicté par les préférences des hommes : chaque homme h fait des propositions par ordre décroissant de ses préférences.
- le choix entre une ancienne liaison h' — f et une nouvelle h — f est dicté par les préférences des femmes (ici, de f).

Il n'est pas évident a priori que cela permettra d'obtenir un c.p.s. !

Une certaine difficulté pédagogique

Il me semble (possible ? mais) difficile d'amener les élèves à découvrir eux-mêmes la définition exacte de l'algorithme.

Il est probablement plus raisonnable de la donner, après en avoir expliqué les grandes lignes.

L'algorithme de Gale et Shapley

L'algorithme s'écrit cette fois naturellement sous forme itérative :

tant que
  il existe un homme libre et qui n'a pas
  demandé toutes les femmes en mariage
faire
  soit h un tel homme
  soit f la préférée de h parmi les femmes
    qu'il n'a pas encore demandées en mariage
  si f est libre **alors**
    h et f se fiancent
  sinon
    soit h' l'actuel fiancé de f
    si f préfère h à h' **alors**
      f quitte h' (qui redevient libre) et se fiance avec h

Non-déterminisme

Cet algorithme est non déterministe : l'instruction « **soit** *h* un tel homme » suppose un choix arbitraire.

Plusieurs niveaux d'abstraction

Cet algorithme est formulé de façon relativement claire, mais :

- très abstraite, en langage naturel stylisé ;
- un peu elliptique : les variables ne sont pas nommées, et les instructions qui mettent à jour ces variables sont omises.

En un sens, c'est bien, mais il faut s'assurer que l'élève comprend les instructions précises, exécutables sous-jacentes.

Reformulons l'algorithme dans un langage plus rigoureux, mais encore très abstrait, où l'on manipule des ensembles.

Une version exécutable

Les variables c et *proposed* stockent un sous-ensemble de $\mathcal{H} \times \mathcal{F}$.

$c \leftarrow \emptyset$
proposed $\leftarrow \emptyset$
tant que $\exists h, f, \quad h \notin \mathrm{dom}(c) \wedge h — f \notin \mathit{proposed}$ **faire**
  soit $h \in \neg\mathrm{dom}(c) \cap \mathrm{dom}(\neg \mathit{proposed})$
  soit $f = \min_{\leq_h}((\neg \mathit{proposed})(h))$
  si $f \notin \mathrm{img}(c)$ **alors**
    $c \leftarrow c \cup \{h — f\}$
  sinon
    soit $h' \in c^{-1}(f)$
    si $h \leq_f h'$ **alors**
      $c \leftarrow c \setminus \{h' — f\} \cup \{h — f\}$
  proposed $\leftarrow \mathit{proposed} \cup \{h — f\}$
renvoyer c

$\neg$ dénote le complément, relatif à $\mathcal{H}$, $\mathcal{F}$, ou $\mathcal{H} \times \mathcal{F}$, selon le contexte.

Un langage ensembliste

Quel est ce langage de programmation ?

Il offre :

- la notion d'ensemble ;
- les opérations « vide », « singleton », « union », « complément », etc. ;
- les opérations de test à vide, de choix d'un élément dans un ensemble non vide, de choix d'un élément minimal (pour un certain ordre) dans un ensemble non vide ;
- la notion de relation, considérée comme un ensemble de paires ;
- les opérations d'image et de pré-image d'un élément ou d'un ensemble à travers une relation.

Ça existe, ce langage ?

Un langage ensembliste

Ça pourrait exister, car ce langage a un sens : il est exécutable.

Ça a existé (voir SETL).

Les « vrais » langages sont de plus bas niveau car un tel « langage ensembliste » est très difficile à compiler efficacement.

Questions

On peut poser de nombreuses à propos de cet algorithme :

- s'exécute-t-il sans erreur ?
- termine-t-il ? si oui, en combien d'itérations de sa boucle principale ?
- produit-il toujours un couplage ? parfait ? stable ?
- peut-on caractériser le c.p.s. qui est choisi ?
- comment le traduit-on dans un langage de plus bas niveau ?
- quelle est alors sa complexité en temps et en espace ?

En bref, sûreté, terminaison, correction, complexité.

Sûreté

Quelles erreurs pourrait-on rencontrer pendant l'exécution ?

- tenter d'exécuter « **soit** $s \in S$ » alors que l'ensemble S est vide.

Dans un langage de plus bas niveau, l'accès aux tableaux serait une source d'erreurs analogue.

Sûreté

Les définitions de h, f, et h' pourraient en principe échouer.

$c \leftarrow \emptyset$
proposed $\leftarrow \emptyset$
tant que $\exists h, f, \quad h \notin \mathrm{dom}(c) \wedge h - f \notin \mathit{proposed}$ **faire**
  soit $h \in \neg\mathrm{dom}(c) \cap \mathrm{dom}(\neg \mathit{proposed})$ — pourrait échouer
  soit $f = \min_{\leq_h}((\neg \mathit{proposed})(h))$ — pourrait échouer
  si $f \notin \mathrm{img}(c)$ **alors**
    $c \leftarrow c \cup \{h - f\}$
  sinon
    soit $h' \in c^{-1}(f)$ — pourrait échouer
    si $h \leq_f h'$ **alors**
      $c \leftarrow c \setminus \{h' - f\} \cup \{h - f\}$
  $\mathit{proposed} \leftarrow \mathit{proposed} \cup \{h - f\}$
renvoyer c

On vérifie à chaque fois qu'il s'agit d'un choix dans un ensemble non vide.

Terminaison

Chaque paire h — f est considérée au plus une fois.

En termes plus techniques, l'ensemble *proposed* croît strictement à chaque itération, et reste un sous-ensemble de $\mathcal{H} \times \mathcal{F}$.

L'algorithme termine donc en n^2 itérations au plus.

Terminaison

$c \leftarrow \emptyset$
proposed $\leftarrow \emptyset$
tant que $\exists h, f, \quad h \notin \mathrm{dom}(c) \wedge h — f \notin \mathit{proposed}$ **faire**
  soit $h \in \neg\mathrm{dom}(c) \cap \mathrm{dom}(\neg\mathit{proposed})$
  soit $f = \min_{\leq_h}((\neg\mathit{proposed})(h))$
  si $f \notin \mathrm{img}(c)$ **alors**
    $c \leftarrow c \cup \{h — f\}$
  sinon
    soit $h' \in c^{-1}(f)$
    si $h \leq_f h'$ **alors**
      $c \leftarrow c \setminus \{h' — f\} \cup \{h — f\}$
  — à démontrer : $h — f \notin \mathit{proposed}$
  $\mathit{proposed} \leftarrow \mathit{proposed} \cup \{h — f\}$
renvoyer c

L'algorithme produit un couplage

Le fait que c est un couplage (i.e., est bijectif) est un invariant de boucle :

- juste avant la boucle, c est vide, donc un couplage.
- si à l'entrée du corps de la boucle c est un couplage, alors à la sortie du corps, il l'est toujours.

On en déduit (par récurrence) que, quel que soit le nombre d'itérations, à la sortie de la boucle, c est un couplage.

L'algorithme produit un couplage

$c \leftarrow \emptyset$
proposed $\leftarrow \emptyset$
— à démontrer : c est un couplage
tant que $\exists h, f, \quad h \notin \mathrm{dom}(c) \wedge h — f \notin \mathit{proposed}$ **faire**
  — supposons que c est un couplage
  soit $h \in \neg\mathrm{dom}(c) \cap \mathrm{dom}(\neg\mathit{proposed})$
  soit $f = \min_{\leq_h}((\neg\mathit{proposed})(h))$
  si $f \notin \mathrm{img}(c)$ **alors**
    $c \leftarrow c \cup \{h — f\}$
  sinon
    soit $h' \in c^{-1}(f)$
    si $h \leq_f h'$ **alors**
      $c \leftarrow c \setminus \{h' — f\} \cup \{h — f\}$

  $\mathit{proposed} \leftarrow \mathit{proposed} \cup \{h — f\}$
  — à démontrer : c est un couplage
renvoyer c

L'algorithme produit un couplage

$c \leftarrow \emptyset$
$\textit{proposed} \leftarrow \emptyset$
— à démontrer : c est un couplage
tant que $\exists h, f, \quad h \notin \mathrm{dom}(c) \wedge h — f \notin \textit{proposed}$ **faire**
  — supposons que c est un couplage
  soit $h \in \neg\mathrm{dom}(c) \cap \mathrm{dom}(\neg\textit{proposed})$
  soit $f = \min_{\leq_h}((\neg\textit{proposed})(h))$
  si $f \notin \mathrm{img}(c)$ **alors** — $h \notin \mathrm{dom}(c)$, $f \notin \mathrm{img}(c)$
    $c \leftarrow c \cup \{h — f\}$ — deux individus libres se fiancent
  sinon
    soit $h' \in c^{-1}(f)$
    si $h \leq_f h'$ **alors**
      $c \leftarrow c \setminus \{h' — f\} \cup \{h — f\}$

  $\textit{proposed} \leftarrow \textit{proposed} \cup \{h — f\}$
  — à démontrer : c est un couplage
renvoyer c

L'algorithme produit un couplage

$c \leftarrow \emptyset$
$\mathit{proposed} \leftarrow \emptyset$
— à démontrer : c est un couplage
tant que $\exists h, f, \quad h \notin \mathrm{dom}(c) \wedge h — f \notin \mathit{proposed}$ **faire**
  — supposons que c est un couplage
  soit $h \in \neg\mathrm{dom}(c) \cap \mathrm{dom}(\neg\mathit{proposed})$
  soit $f = \min_{\leq_h}((\neg\mathit{proposed})(h))$
  si $f \notin \mathrm{img}(c)$ **alors** — $h \notin \mathrm{dom}(c)$, $f \notin \mathrm{img}(c)$
    $c \leftarrow c \cup \{h — f\}$ — deux individus libres se fiancent
  sinon
    soit $h' \in c^{-1}(f)$ — $h' — f \in c$
    si $h \leq_f h'$ **alors** — $h \notin \mathrm{dom}(c \setminus \{h' — f\})$, $f \notin \mathrm{img}(c \setminus \{h' — f\})$
      $c \leftarrow c \setminus \{h' — f\} \cup \{h — f\}$
      — f quitte son fiancé h' pour h, qui était libre
  $\mathit{proposed} \leftarrow \mathit{proposed} \cup \{h — f\}$
  — à démontrer : c est un couplage
renvoyer c

L'algorithme produit un couplage parfait

tant que $\exists h, f, \quad h \notin \mathrm{dom}(c) \wedge h — f \notin \mathit{proposed}$ **faire** $\ldots$

Lorsque l'algorithme termine, on sait qu'il n'y a plus d'homme libre ou bien que tous les hommes libres ont fait une proposition à toutes les femmes.

Dans le premier cas, tous les hommes sont fiancés : le couplage est parfait.

Dans le second cas, toutes les femmes ont reçu une proposition. Or, une femme qui a reçu une proposition reste toujours fiancée. Donc, toutes les femmes sont fiancées : le couplage est parfait.

L'algorithme produit un couplage parfait

En termes plus techniques, on souhaite procéder ainsi :

- montrer que $\mathrm{img}(\mathit{proposed}) \subseteq \mathrm{img}(c)$ est un invariant de boucle : une femme qui a reçu une proposition reste fiancée.
- indépendamment de cela, montrer que, à la sortie de la boucle, soit $\mathrm{dom}(c)$ est plein (tous les hommes sont fiancés), soit $\mathrm{img}(\mathit{proposed})$ est plein (toutes les femmes ont reçu une proposition) ;
- en combinant ces deux points, déduire que, à la sortie de la boucle, c est un couplage parfait.

Qui a reçu une proposition reste fiancée

$c \leftarrow \emptyset$
proposed $\leftarrow \emptyset$
— à démontrer : $\mathrm{img}(\mathit{proposed}) \subseteq \mathrm{img}(c)$
tant que $\exists h, f, \quad h \notin \mathrm{dom}(c) \wedge h — f \notin \mathit{proposed}$ **faire**
  — supposons $\mathrm{img}(\mathit{proposed}) \subseteq \mathrm{img}(c)$
  soit $h \in \neg\mathrm{dom}(c) \cap \mathrm{dom}(\neg\mathit{proposed})$
  soit $f = \min_{\leq_h}((\neg\mathit{proposed})(h))$
  si $f \notin \mathrm{img}(c)$ **alors**
    $c \leftarrow c \cup \{h — f\}$

  sinon
    soit $h' \in c^{-1}(f)$
    si $h \leq_f h'$ **alors**
      $c \leftarrow c \setminus \{h' — f\} \cup \{h — f\}$
  $\mathit{proposed} \leftarrow \mathit{proposed} \cup \{h — f\}$
  — à démontrer : $\mathrm{img}(\mathit{proposed}) \subseteq \mathrm{img}(c)$
renvoyer c

Qui a reçu une proposition reste fiancée

$c \leftarrow \emptyset$
$proposed \leftarrow \emptyset$
— à démontrer : $\mathrm{img}(proposed) \subseteq \mathrm{img}(c)$
tant que $\exists h, f, \quad h \notin \mathrm{dom}(c) \wedge h — f \notin proposed$ **faire**
  — supposons $\mathrm{img}(proposed) \subseteq \mathrm{img}(c)$
  soit $h \in \neg\mathrm{dom}(c) \cap \mathrm{dom}(\neg proposed)$
  soit $f = \min_{\leq_h}((\neg proposed)(h))$
  si $f \notin \mathrm{img}(c)$ **alors** — $f \notin \mathrm{img}(c)$, $f \notin \mathrm{img}(proposed)$
    $c \leftarrow c \cup \{h — f\}$
    — f reçoit une première proposition, et devient fiancée
  sinon
    soit $h' \in c^{-1}(f)$
    si $h \leq_f h'$ **alors**
      $c \leftarrow c \setminus \{h' — f\} \cup \{h — f\}$
  $proposed \leftarrow proposed \cup \{h — f\}$
  — à démontrer : $\mathrm{img}(proposed) \subseteq \mathrm{img}(c)$
renvoyer c

Qui a reçu une proposition reste fiancée

$c \leftarrow \emptyset$
proposed $\leftarrow \emptyset$
— à démontrer : $\mathrm{img}(\mathit{proposed}) \subseteq \mathrm{img}(c)$
tant que $\exists h, f, \quad h \notin \mathrm{dom}(c) \wedge h — f \notin \mathit{proposed}$ **faire**
  — supposons $\mathrm{img}(\mathit{proposed}) \subseteq \mathrm{img}(c)$
  soit $h \in \neg\mathrm{dom}(c) \cap \mathrm{dom}(\neg\mathit{proposed})$
  soit $f = \min_{\leq_h}((\neg\mathit{proposed})(h))$
  si $f \notin \mathrm{img}(c)$ **alors** — $f \notin \mathrm{img}(c)$, $f \notin \mathrm{img}(\mathit{proposed})$
    $c \leftarrow c \cup \{h — f\}$
    — f reçoit une première proposition, et devient fiancée
  sinon — $f \in \mathrm{img}(c)$
    soit $h' \in c^{-1}(f)$
    si $h \leq_f h'$ **alors**
      $c \leftarrow c \setminus \{h' — f\} \cup \{h — f\}$ — f était fiancée et le reste
  $\mathit{proposed} \leftarrow \mathit{proposed} \cup \{h — f\}$
  — à démontrer : $\mathrm{img}(\mathit{proposed}) \subseteq \mathrm{img}(c)$
renvoyer c

Qui a reçu une proposition reste fiancée

$c \leftarrow \emptyset$
$\mathit{proposed} \leftarrow \emptyset$
— à démontrer : $\mathrm{img}(\mathit{proposed}) \subseteq \mathrm{img}(c)$
tant que $\exists h, f, \quad h \notin \mathrm{dom}(c) \wedge h — f \notin \mathit{proposed}$ **faire**
  — supposons $\mathrm{img}(\mathit{proposed}) \subseteq \mathrm{img}(c)$
  soit $h \in \neg\mathrm{dom}(c) \cap \mathrm{dom}(\neg\mathit{proposed})$
  soit $f = \min_{\leq_h}((\neg\mathit{proposed})(h))$
  si $f \notin \mathrm{img}(c)$ **alors** — $f \notin \mathrm{img}(c)$, $f \notin \mathrm{img}(\mathit{proposed})$
    $c \leftarrow c \cup \{h — f\}$
    — f reçoit une première proposition, et devient fiancée
  sinon — $f \in \mathrm{img}(c)$, $f \in \mathrm{img}(\mathit{proposed})$?
    soit $h' \in c^{-1}(f)$
    si $h \leq_f h'$ **alors**
      $c \leftarrow c \setminus \{h' — f\} \cup \{h — f\}$ — f était fiancée et le reste
  $\mathit{proposed} \leftarrow \mathit{proposed} \cup \{h — f\}$
  — à démontrer : $\mathrm{img}(\mathit{proposed}) \subseteq \mathrm{img}(c)$
renvoyer c

Une vérité n'est pas forcément un invariant

On tombe sur un petit os : pour montrer que $\mathrm{img}(\mathit{proposed}) \subseteq \mathrm{img}(c)$ est préservée, nous avons besoin de la réciproque, $\mathrm{img}(c) \subseteq \mathrm{img}(\mathit{proposed})$: toute fiancée a reçu une proposition.

La propriété $\mathrm{img}(\mathit{proposed}) \subseteq \mathrm{img}(c)$ est vraie (dans l'absolu) mais n'est pas (à elle seule) un invariant de boucle.

Il faut la renforcer. Nous allons montrer que $\mathrm{img}(\mathit{proposed}) = \mathrm{img}(c)$ est un invariant de boucle.

Qui a reçu une avance est fiancée, et vice-versa

$c \leftarrow \emptyset$
proposed $\leftarrow \emptyset$
— à démontrer : $\mathrm{img}(\mathit{proposed}) = \mathrm{img}(c)$
tant que $\exists h, f, \quad h \notin \mathrm{dom}(c) \wedge h — f \notin \mathit{proposed}$ **faire**
  — supposons $\mathrm{img}(\mathit{proposed}) = \mathrm{img}(c)$
  soit $h \in \neg\mathrm{dom}(c) \cap \mathrm{dom}(\neg\mathit{proposed})$
  soit $f = \min_{\leq_h}((\neg\mathit{proposed})(h))$
  si $f \notin \mathrm{img}(c)$ **alors**
    $c \leftarrow c \cup \{h — f\}$
  sinon
    soit $h' \in c^{-1}(f)$
    si $h \leq_f h'$ **alors**
      $c \leftarrow c \setminus \{h' — f\} \cup \{h — f\}$
  proposed $\leftarrow \mathit{proposed} \cup \{h — f\}$
  — à démontrer : $\mathrm{img}(\mathit{proposed}) = \mathrm{img}(c)$
renvoyer c

Qui a reçu une avance est fiancée, et vice-versa

$c \leftarrow \emptyset$
proposed $\leftarrow \emptyset$
— à démontrer : $\mathrm{img}(\mathit{proposed}) = \mathrm{img}(c)$
tant que $\exists h, f, \quad h \notin \mathrm{dom}(c) \wedge h — f \notin \mathit{proposed}$ **faire**
  — supposons $\mathrm{img}(\mathit{proposed}) = \mathrm{img}(c)$
  soit $h \in \neg\mathrm{dom}(c) \cap \mathrm{dom}(\neg\mathit{proposed})$
  soit $f = \min_{\leq_h}((\neg\mathit{proposed})(h))$
  si $f \notin \mathrm{img}(c)$ **alors** — $f \notin \mathrm{img}(c)$, $f \notin \mathrm{img}(\mathit{proposed})$
    $c \leftarrow c \cup \{h — f\}$
  sinon
    soit $h' \in c^{-1}(f)$
    si $h \leq_f h'$ **alors**
      $c \leftarrow c \setminus \{h' — f\} \cup \{h — f\}$
  proposed $\leftarrow$ *proposed* $\cup \{h — f\}$
  — à démontrer : $\mathrm{img}(\mathit{proposed}) = \mathrm{img}(c)$
renvoyer c

Qui a reçu une avance est fiancée, et vice-versa

$c \leftarrow \emptyset$
proposed $\leftarrow \emptyset$
— à démontrer : $\mathrm{img}(\mathit{proposed}) = \mathrm{img}(c)$
tant que $\exists h, f, \quad h \notin \mathrm{dom}(c) \wedge h — f \notin \mathit{proposed}$ **faire**
  — supposons $\mathrm{img}(\mathit{proposed}) = \mathrm{img}(c)$
  soit $h \in \neg\mathrm{dom}(c) \cap \mathrm{dom}(\neg\mathit{proposed})$
  soit $f = \min_{\leq_h}((\neg\mathit{proposed})(h))$
  si $f \notin \mathrm{img}(c)$ **alors** — $f \notin \mathrm{img}(c)$, $f \notin \mathrm{img}(\mathit{proposed})$
    $c \leftarrow c \cup \{h — f\}$
  sinon — $f \in \mathrm{img}(c)$, $f \in \mathrm{img}(\mathit{proposed})$
    soit $h' \in c^{-1}(f)$
    si $h \leq_f h'$ **alors**
      $c \leftarrow c \setminus \{h' — f\} \cup \{h — f\}$
  $\mathit{proposed} \leftarrow \mathit{proposed} \cup \{h — f\}$
  — à démontrer : $\mathrm{img}(\mathit{proposed}) = \mathrm{img}(c)$
renvoyer c

Une simplification

Au passage, nous avons démontré que :

s'il reste un homme h libre,
alors il reste une femme f libre, (puisque c est un couplage)
et f n'a reçu aucune proposition, (puisque $\mathrm{img}(\mathit{proposed}) = \mathrm{img}(c)$)
donc $h — f \notin \mathit{proposed}$.

De ce fait, la condition de la boucle :

$$\textbf{tant que } \exists h, f, \quad h \notin \mathrm{dom}(c) \wedge h — f \notin \mathit{proposed} \textbf{ faire } \ldots$$

peut être simplifiée sans modifier le comportement de l'algorithme :

$$\textbf{tant que } \exists h, \quad h \notin \mathrm{dom}(c) \textbf{ faire } \ldots$$

L'algorithme produit un couplage stable

Comment montrer que c est stable ?

Pour montrer l'absence d'instabilité, supposons :

- $h_1 — f_1 \in c$;
- $h_2 — f_2 \in c$;
- $f_2 \leq_{h_1} f_1$ (i.e., h_1 préfère f_2 à sa fiancée f_1).

Nous devons démontrer :

- $h_2 \leq_{f_2} h_1$ (i.e., f_2 préfère son fiancé actuel).

Ainsi, f_2 ne cèdera pas aux éventuelles avances de h_1.

L'algorithme produit un couplage stable

Nous avons $h_1 — f_1 \in c$ (1) et $h_2 — f_2 \in c$ (2) et $f_2 \leq_{h_1} f_1$ (3).

Dans ses grandes lignes, la preuve est simple :

	$h_1 — f_1 \in c$	(1)
donc	$h_1 — f_1 \in \mathit{proposed}$	car $c \subseteq \mathit{proposed}$ (A)
donc	$h_1 — f_2 \in \mathit{proposed}$	d'après (3) et car un homme (ici h_1) émet des propositions dans l'ordre de ses préférences (B)
donc	$h_2 \leq_{f_2} h_1$	d'après (2) et car le fiancé actuel d'une femme (ici f_2) lui plaît au moins autant que les précédents (C)

Les points A, B, C peuvent sembler évidents. Techniquement, on peut les formuler en termes d'ensembles et montrer que ce sont eux aussi des invariants de boucle...

Trois invariants auxiliaires

On peut formuler et vérifier ces trois invariants auxiliaires :

A. Toute fiançaille résulte d'une proposition.

$$c \subseteq \mathit{proposed}$$

B. Un homme émet des propositions dans l'ordre de ses préférences.

$$\forall h, f_1, f_2, \quad h — f_1 \in \mathit{proposed} \wedge f_2 \leq_h f_1 \Rightarrow h — f_2 \in \mathit{proposed}$$

C. Le fiancé d'une femme lui plaît au moins autant que les précédents.

$$\forall h_1, h_2, f, \quad h_1 — f \in \mathit{proposed} \wedge h_2 — f \in c \Rightarrow h_2 \leq_f h_1$$

À propos du raisonnement temporel

Le discours intuitif emploie présent et passé : « si h_1 et f_1 sont fiancés, alors h_1 a forcément fait une proposition à f_2 auparavant... ».

Dans le discours formel, tout s'exprime au présent : « $h_1 — f_1 \in c$ implique $h_1 — f_2 \in \mathit{proposed}$ ». Tout le raisonnement est fondé sur des invariants de boucle.

En théorie, cette technique fondée sur les invariants est suffisante.

D'un point de vue pédagogique, rester à un niveau intuitif peut parfois être préférable !

Conclusion intermédiaire

Nous avons donc démontré que l'algorithme termine et produit un c.p.s..

En d'autres termes, nous avons démontré (de façon effective) qu'il existe toujours un c.p.s., ce qui n'était pas évident.

Un non-déterminisme seulement apparent

Soit $\mathcal{S}$ l'ensemble de tous les couplages parfaits stables. $\mathcal{S}$ est non vide.

L'algorithme, non déterministe, produit un élément de $\mathcal{S}$ *a priori* quelconque.

En réalité, nous allons constater que le c.p.s. obtenu est toujours le même.

Cet algorithme non déterministe satisfait une spécification déterministe.

Que peut espérer de mieux un homme ?

Du point de vue d'un homme h, la femme la plus désirable qu'il peut espérer obtenir dans un couplage parfait stable est :

$$S^\star(h) = \min_{\leq_h}\{f \mid \exists c \in \mathcal{S}, \quad h - f \in c\}$$

$S^\star$ est une fonction totale de $\mathcal{H}$ dans $\mathcal{F}$.

Elle est bien définie, car $\mathcal{S}$ est non vide.

$S^\star(h)$ est la meilleure partenaire réalisable de h.

Étude de la structure de $S^\star$

Lemme (préliminaire)

$h \neq h'$ et $S^\star(h) = f \leq_{h'} S^\star(h')$ impliquent $h <_f h'$.

Démonstration.

Supposons $h \neq h'$ et $S^\star(h) = f \leq_{h'} S^\star(h')$.

Par définition de $S^\star$, il existe un c.p.s. c contenant h — f.

Ce couplage contient alors une autre paire h' — f', où $f \neq f'$.

Par définition de $S^\star$, on a $S^\star(h') \leq_{h'} f'$, d'où (par transitivité) $f <_{h'} f'$.

Par stabilité du couplage c, il s'ensuit $h <_f h'$. □

$S^\star$ est stable

Du lemme précédent, il découle immédiatement :

Lemme ($S^\star$ est stable)

$S^\star(h) <_{h'} S^\star(h')$ implique $h <_{S^\star(h)} h'$.

Démonstration.

Supposons $S^\star(h) <_{h'} S^\star(h')$.

Il en découle $h \neq h'$.

Le lemme précédent s'applique et donne le résultat. $\square$

$S^\star$ est un couplage parfait

Du lemme préliminaire, il découle également :

Lemme ($S^\star$ est un couplage parfait)

$S^\star$ est injectif.

Démonstration.

Supposons $S^\star(h) = f = S^\star(h')$, où $h \neq h'$.

Le lemme préliminaire donne $h <_f h'$.

Par symétrie, il donne également $h' <_f h$.

Contradiction. □

L'algorithme est globalement déterministe

Nous allons voir que le couplage produit par l'algorithme est toujours $S^\star$.

C'est étonnant ! L'algorithme est en fait globalement déterministe.

L'algorithme favorise les hommes

Chaque homme est donc sûr d'obtenir sa meilleure partenaire réalisable.

L'algorithme est asymétrique : il favorise les hommes.

Bien sûr, on peut le renverser pour favoriser les femmes.

Idée

Comment montrer que le couplage c produit par l'algorithme est $S^\star$?

Montrons qu'un homme h ne sera jamais amené à faire une proposition à une femme située au-delà de $S^\star(h)$ dans sa liste de préférences. Donc, la femme f obtenue finalement par h vérifiera $f \leq_h S^\star(h)$.

Par ailleurs, puisque c est un c.p.s. et par définition de $S^\star$, cette femme f vérifiera nécessairement $S^\star(h) \leq_h f$.

D'où le résultat.

Un double invariant

Nous sommes amenés à formuler et établir un double invariant :

- si h et f sont fiancés, alors h préfère f à $S^\star(h)$, au sens large ;
- si h est libre, et si h a fait par le passé une proposition à f, alors h préfère f à $S^\star(h)$, au sens strict.

La première propriété est celle qui nous intéresse in fine.

La seconde est nécessaire pour que le tout forme un invariant de boucle.

Un double invariant

On peut le formuler de façon formelle :

- $\forall h, \quad h \in \mathrm{dom}(c) \Rightarrow c(h) \leq_h S^\star(h)$;
- $\forall h, \quad h \notin \mathrm{dom}(c) \Rightarrow \mathit{proposed}(h) <_h S^\star(h)$.

Vérifions que cet invariant est préservé

$c \leftarrow \emptyset$
proposed $\leftarrow \emptyset$
tant que $\mathrm{dom}(c) \neq \emptyset$ **faire**
  — supposons l'invariant
  soit $h \notin \mathrm{dom}(c)$
  soit $f = \min_{\leq_h}((\neg proposed)(h))$
  — $f \leq_h S^\star(h)$ découle de l'invariant
  si $f \notin \mathrm{img}(c)$ **alors**
    $c \leftarrow c \cup \{h — f\}$
  sinon
    soit $h' \in c^{-1}(f)$
    — $f \leq_{h'} S^\star(h')$ découle de l'invariant
    si $h \leq_f h'$ **alors**
      $c \leftarrow c \setminus \{h' — f\} \cup \{h — f\}$
      — à démontrer : $f \neq S^\star(h')$
    sinon
      — à démontrer : $f \neq S^\star(h)$
  $proposed \leftarrow proposed \cup \{h — f\}$
renvoyer c

Vérifions que cet invariant est préservé

Intuitivement, les deux points à vérifier sont de même nature.

Lorsqu'un homme (h ou h', selon le cas) constate qu'il devra aller au-delà de f, il faut vérifier que f n'est pas la meilleure partenaire réalisable de cet homme.

Ainsi, un homme ne dépasse jamais sa meilleure partenaire réalisable.

Vérifions que cet invariant est préservé

Les deux points à vérifier sont conséquence d'une même propriété :

$$f \leq_h S^\star(h) \textit{ et } h <_f h' \textit{ impliquent } f \neq S^\star(h').$$

Pour la démontrer, on la réécrit sous la forme équivalente :

$$S^\star(h') \leq_h S^\star(h) \textit{ implique } h' \leq_{S^\star(h')} h.$$

On constate alors que c'est exactement la stabilité de $S^\star$.

Choix des structures de données

> *Algorithms + Data Structures = Programs*
> *– Niklaus Wirth*

Nous avons écrit l'algorithme dans un langage « ensembliste ».

Nous devons le traduire (manuellement) vers un langage de plus bas niveau (ici, Java) en choisissant des structures de données adaptées.

Choix des structures de données

Nous devons choisir comment représenter $\mathcal{H}$, $\mathcal{F}$, $\leq_h$, $\leq_f$, c, et *proposed*.

$c \leftarrow \emptyset$
proposed $\leftarrow \emptyset$
tant que $\mathrm{dom}(c) \neq \emptyset$ **faire**
  soit $h \notin \mathrm{dom}(c)$
  soit $f = \min_{\leq_h}((\neg \mathit{proposed})(h))$
  si $f \notin \mathrm{img}(c)$ **alors**
    $c \leftarrow c \cup \{h — f\}$
  sinon
    soit $h' \in c^{-1}(f)$
    si $h \leq_f h'$ **alors**
      $c \leftarrow c \setminus \{h' — f\} \cup \{h — f\}$
  proposed $\leftarrow$ *proposed* $\cup \{h — f\}$
renvoyer c

Ce choix dépend des opérations que l'on doit effectuer efficacement.

Représentation de $\mathcal{H}$ et $\mathcal{F}$

Ces ensembles peuvent être représentés implicitement par leur cardinal n.

Le client fournit donc n, et on considère que $\mathcal{H} = \mathcal{F} = [0, n)$.

```
int n;
```

C'est simple, et cela nous permet ensuite d'employer des tableaux pour associer des informations à un individu.

Si le problème est initialement exprimé en termes d'individus nommés, le client devra les numéroter (de façon arbitraire) avant d'utiliser l'algorithme.

Cela pourra exiger l'emploi d'une table d'association (table de hash, arbre binaire de recherche, ...).

Représentation de $\leq_h$ et $\leq_f$

La relation $\leq_h$ est un ordre total sur $\mathcal{F}$, donc une liste de toutes les femmes, ou une permutation des femmes.

On peut la représenter par un tableau de taille n, dont les éléments sont les éléments de $\mathcal{F}$, rangés par ordre de préférence décroissante.

Il faut un tel tableau pour chaque homme h. Le client fournit donc un tableau de tableaux, ou une matrice à deux dimensions, `menPrefs`.

```
int[][]   menPrefs;
int[][] womenPrefs;
```

Les relations $\leq_f$ sont fournies par le client sous la même forme.

Représentation de $\leq_h$ et $\leq_f$

Ces représentations sont-elles adaptées aux opérations que nous effectuons ?

La relation $\leq_h$ est utilisée pour choisir à qui faire une proposition :

$$\textbf{soit } f = \min_{\leq_h}((\neg \mathit{proposed})(h))$$

Chaque homme procède par ordre décroissant de préférence, donc représenter $\leq_h$ par une liste est adéquat.

Représentation de $\leq_h$ et $\leq_f$

La relation $\leq_f$ est utilisée pour comparer deux hommes :

si $h \leq_f h'$ **alors**

Si $\leq_f$ est représenté par une liste d'hommes, il faut un temps $O(n)$ pour déterminer lequel, parmi h et h', apparaît en premier dans cette liste.

Représentons plutôt $\leq_f$ par un tableau qui à chaque homme h associe son rang dans l'ordre $\leq_f$. Alors, en temps $O(1)$, on pourra obtenir et comparer les rangs de h et h'.

```
int[][] womenRanks = new int[n][];
for (int w = 0; w < n; w++)
  womenRanks[w] = inversePermutation(womenPrefs[w]);
```

Représentation de *proposed*

En principe, *proposed* est un ensemble de paires h — f.

Cependant, nous avons l'invariant B :

$$h \text{ — } f_1 \in \mathit{proposed} \wedge f_2 \leq_h f_1 \Rightarrow h \text{ — } f_2 \in \mathit{proposed}$$

Pour chaque homme h, il suffit donc de mémoriser le rang dans la liste $\leq_h$ de la première femme à qui h n'a pas encore fait de proposition.

```
int[] nextWoman = new int [n];
```

Représentation de *proposed*

Cette représentation est-elle adaptée à nos besoins ?

Oui.

Les deux opérations qui concernent *proposed* :

$$\textbf{soit } f = \min_{\leq_h}((\neg proposed)(h))$$
$$proposed \leftarrow proposed \cup \{h — f\}$$

se traduisent aisément et efficacement :

```
int w = menPrefs[m][nextWoman[m]++];
```

Notons que nous avons déjà démontré que cet accès respecte les bornes du tableau `menPrefs[m]`.

Représentation de c

En principe, c est un ensemble de paires h — f.

Cependant, nous savons que c'est un couplage, donc une fonction partielle injective de $\mathcal{H}$ dans $\mathcal{F}$.

On pourrait donc le représenter, de façon compacte, par un tableau qui à un homme associe sa fiancée (s'il en a une) et une valeur spéciale sinon.

Inversement, c^{-1} est une fonction partielle injective de $\mathcal{F}$ dans $\mathcal{H}$.

Voyons quelle représentation conviendrait le mieux...

Représentation de c

$c \leftarrow \emptyset$
proposed $\leftarrow \emptyset$
tant que $\mathrm{dom}(c) \neq \emptyset$ **faire** — test si domaine vide
  soit $h \notin \mathrm{dom}(c)$ — choix dans le domaine
  soit $f = \min_{\leq_h}((\neg \mathit{proposed})(h))$
  si $f \notin \mathrm{img}(c)$ **alors** — test d'appartenance à l'image
    $c \leftarrow c \cup \{h — f\}$ — mise à jour
  sinon
    soit $h' \in c^{-1}(f)$ — calcul de l'antécédent
    si $h \leq_f h'$ **alors**
      $c \leftarrow c \setminus \{h' — f\} \cup \{h — f\}$ — mise à jour
  $\mathit{proposed} \leftarrow \mathit{proposed} \cup \{h — f\}$
renvoyer c

Représentation de c

Représenter c comme une fonction de $\mathcal{F}$ dans $\mathcal{H}$ facilite le test d'appartenance à l'image et le calcul de l'antécédent.

```
final int NONE = -1;
int[] currentGroom = new int [n];
for (int w = 0; w < n; w++)
  currentGroom[w] = NONE;
```

Le test $f \notin \mathrm{img}(c)$ s'écrit `currentGroom[w] == NONE`.

L'instruction **soit** $h' \in c^{-1}(f)$ devient `int otherMan = currentGroom[w]`.

Représentation de c

Cependant, `currentGroom` ne permet pas de tester efficacement si $\mathrm{dom}(c)$ est vide, ni d'en choisir un élément.

Ceci nous conduit à utiliser une autre structure de données pour représenter l'ensemble $\mathrm{dom}(c)$.

Nous aurons ainsi une représentation redondante de c.

Représentation de *c*

Cette structure doit permettre le test si vide, le choix d'un élément arbitraire (et son retrait), et l'insertion.

On appelle parfois cela un « bag ».

Comme cet ensemble contient au maximum *n* éléments, on peut le représenter comme une pile stockée dans un tableau.

```
int[] freeMen = new int [n];
int   freeMenTop = 0;
for (int m = 0; m < n; m++)
  freeMen[m] = m;
```

Représentation redondante de c

Pour consulter c, on utilise soit `currentGroom` soit `freeMen`.

Pour modifier c, on met à jour ces deux structures.

Le code

Hormis les instructions d'initialisation déjà présentées, le code s'écrit :

```
while (freeMenTop < n) {
  int m = freeMen[freeMenTop];
  int w = menPrefs[m][nextWoman[m]++];
  int otherMan = currentGroom[w];
  if (otherMan == NONE) {
    freeMenTop++;
    currentGroom[w] = m;
  }
  else if (womenRanks[w][m] < womenRanks[w][otherMan]) {
    freeMen[freeMenTop] = otherMan;
    currentGroom[w] = m;
  }
}
return currentGroom;
```

On choisit de renvoyer c^{-1}, mais cela se modifie aisément.

Complexité

Le corps de la boucle contient :

- des opérations élémentaires (additions, comparaisons) ;
- des accès à la mémoire (accès aux tableaux) ;
- des sauts conditionnels (**si**/**alors**/**sinon**).

Il est donc exécuté en temps constant : $O(1)$.

Complexité

La complexité en temps de l'algorithme de Gale et Shapley, pour des structures de données bien choisies, est donc $O(n^2)$ dans le cas le pire.

C'est optimal, car la description du problème a une taille $O(n^2)$, et il faut bien la lire (argument informel).

La complexité en espace est également $O(n^2)$.

Emballage

L'algorithme complet prend la forme d'une fonction Java :

```
int[] constructStableMatching (
  int n,
  int[][] menPrefs,
  int[][] womenPrefs
)
```

accompagné idéalement d'un commentaire qui spécifie précisément ce qui est attendu et ce qui est produit.

L'activité algorithmique

Elle consiste en principe à :

- présenter et analyser un problème ;
- proposer un algorithme pour le résoudre ;
- poser les questions de la sûreté, terminaison, correction, complexité.

Quelques suggestions d'activités

Que faire faire aux élèves avant de programmer ?

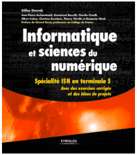
- bien sûr, définir l'algorithme et l'étudier, comme on vient de le voir ;
- simuler son comportement, pas à pas, au tableau.

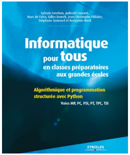
Et pendant la phase de programmation ?

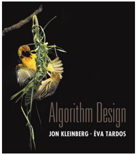
- fournir un code de test qui explique en quoi le résultat est incorrect ;
- ou bien demander aux élèves de l'écrire ! ce qui les fera réfléchir.
- demander d'écrire des assertions qui vérifient pendant l'exécution la validité des invariants supposés ;
- demander que le programme affiche un journal de ses actions.

Références bibliographiques

Quelques livres :







Un site Web :

Science Info Lycée