

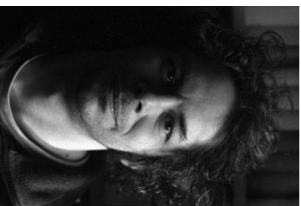
INF431

cette année

## Algorithmique et Programmation

*Conception et mise en œuvre d'algorithmes.*

Benjamin Werner



François Potier



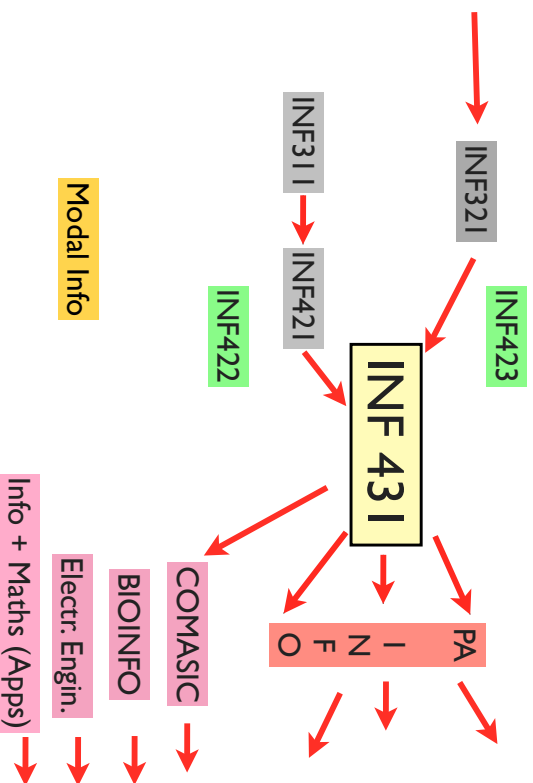
Pourquoi ce cours ?

- ▶ L'occasion de programmer
- ▶ Importance de la culture algorithmique

Les algorithmes

- ▶ Un concept fondamental
- ▶ Un outil mathématique
- ▶ L'abstraction derrière les programmes informatiques

Ils sont partout



## Un exemple : le simplexe

$$\begin{array}{l} \text{Minimiser} \\ \text{Pourvu que} \\ \text{et} \end{array} \quad \begin{array}{l} c^T x \\ Ax = B \\ l \leq x \leq u \end{array}$$

Programmation linéaire ; nombreuses applications.

En 28 ans :

- ▶ algorithme :  $2360 \times$  plus vite
- ▶ machines :  $800 \times$
- total :  $\simeq 2.000.000 \times$

Traité en troisième année : Programmation par contraintes

source : *Gilles Kahn et Bob Bixby*; [interstices.info](http://interstices.info)



## Plan du cours

Amphis B. Werner			
1	1 <sup>er</sup> février	Présentation ; problème des mariages stables	PC
2	8 février	Programmation orienté-objet en Java	TD
3	15 février	Programmation récursive, raisonnement récursif	TD
4	29 février	Diviser pour régner	PC
5	7 mars	Structures mutables (+introduction graphes)	TD
6	14 mars	Graphes	PC
7	21 mars	Algorithmes gloutons (1) : algorithme de Dijkstra	TD
8	28 mars	Algorithmes gloutons (2) : arbres couvrants minimaux...	PC
9	4 avril	Programmation Dynamique	PC
Amphis F. Potier			
10	11 avril	Le Test	TD
11	2 mai	Preuves de programmes	PC/TD
12	9 mai	Exploration	PC
13	23 mai	Interprétation de programmes	TD
14	30 mai	Analyse syntaxique (1)	PC
15	6 juin	Analyse syntaxique (2)	PC
16	13 juin	Programmation concurrente (1)	TD
17	20 juin	Programmation concurrente (2)	TD
18	27 juin	Programmation concurrente (3)	PC



## Organisation du cours

<http://www.enseignement.polytechnique.fr/informatique/INF431/>  
("INF431" sur votre moteur de recherche préféré)

- ▶ 18 blocs : amphi + TD (machine) ou PC (papier)
- ▶ Langage : Java
- ▶ CC1 le 25 avril (2h)
- ▶ CC2 le 4 juillet (3h)
- ▶ Projet Informatique : sujets mi-février, choix en mars, rendu fin mai, soutenances en juin.
- ▶ Poly : tout nouveau. Version complète dans 3 semaines. Chapitres à part en attendant.



## Calcul de la note

Note classante :  $CC = CC1/3 + 2 \times CC2/3$

Note de module :

$$(P1 + 2 \times CC)/3 + (TD + DM)$$

avec  $(TD + DM) \in [-1; 1]$



## Le projet informatique

Important, apprécié, même si c'est du travail

Par binômes a priori. Environ 40h.

Sujets bientôt. Dates : choix début mars, rendu fin mai, soutenance en juin.

Choisissez bien votre sujet. Sujet perso possible si vous trouvez un enseignant qui donne son accord pour l'encadrer.

Pas de validation du cours sans projet

Pas de plagiat !! (graves ennuis)



## Existence et calcul effectif : un exemple

**Lemme** Il existe  $(a, b) \in \mathbb{R}$ , avec :

- ▶  $a$  et  $b$  irrationnels
- ▶  $a^b \in \mathbb{Q}$ .

**Démonstration**

$\sqrt{2} \notin \mathbb{Q}$  (OK, c'est connu)

- ▶ Si  $\sqrt{2}^{\sqrt{2}} \in \mathbb{Q}$  on prend  $a = b = \sqrt{2}$ .
- ▶ Si  $\sqrt{2}^{\sqrt{2}} \notin \mathbb{Q}$ , on prend :  
 $a = \sqrt{2}^{\sqrt{2}}$  et  $b = \sqrt{2}$ . En effet alors :

$$a^b = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{\sqrt{2} \times \sqrt{2}} = \sqrt{2}^2 = 2 \in \mathbb{Q}$$

Ok, but what's the point ?

Quels sont les  $a$  et  $b$  ?

On ne sait pas : il n'y a pas d'algorithme dans cette preuve.

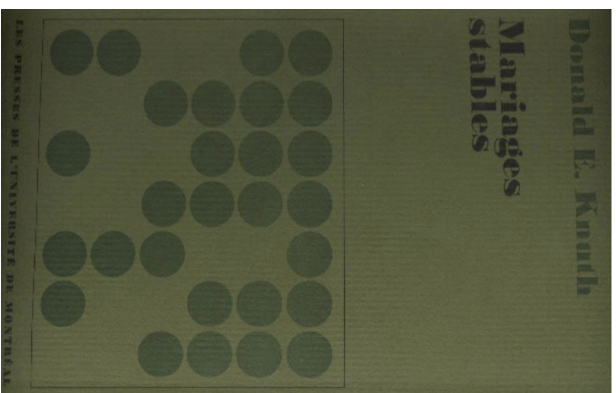


Les algorithmes

Le problème des mariages stables

Stable matchings problem





## Le Problème

Deux ensembles  $\mathcal{G}$  et  $\mathcal{F}$  de cardinal  $n$ .

**Couplage** :  $\mathcal{C} \subset \mathcal{G} \times \mathcal{F}$ , chaque  $f$  et chaque  $g$  apparaît dans au plus un élément de  $\mathcal{C}$ .

**Couplage parfait** : chaque élément de  $\mathcal{F}$  (et donc chaque élément de  $\mathcal{G}$ ) apparaît dans exactement un élément de  $\mathcal{C}$ .

Et l'amour dans tout ça ?

Chaque  $g \in \mathcal{G}$  classe les éléments de  $\mathcal{F}$  totalement.

Chaque  $f \in \mathcal{F}$  classe les éléments de  $\mathcal{G}$  totalement.

notation :  $f_1 \succ_g f_2$        $g$  préfère  $f_1$  à  $f_2$

## Stabilité

$(g_1, f_1) \in \mathcal{C}$     $(g_2, f_2) \in \mathcal{C}$

instable si  $f_1$  préfère  $g_2$  à  $f_1$  et  $g_2$  préfère  $f_1$  à  $f_2$ .

Si  $\mathcal{C}$  est un couplage parfait sans instabilité, c'est un **couplage stable**

- ▶ Existe-t-il un couplage stable ?
- ▶ Comment le calculer ?
- ▶ Comment le calculer efficacement ?

AVANT-PROPOS

Ces dix dernières années ont vu le développement rapide d'un domaine qui a reçu le nom d'"informatique". À mon avis, il conviendrait d'appeler "Algorithmique" cette discipline dont l'objet principal n'est pas l'étude de l'information elle-même, mais celle des processus de traitement de l'information, ou algorithmes. Quoi qu'il en soit, sous un nom ou un autre, ce domaine s'est avéré passionnant à plus d'un titre.

Le but de cet ouvrage est d'introduire le lecteur à l'Analyse des Algorithmes, en se basant sur des exemples plutôt qu'en faisant le tour des principaux résultats théoriques. Cette forme de présentation saura, je l'espère, lui donner une idée des méthodes utilisées dans ce domaine, et illustrer les rapports étroits qu'il entretient avec de nombreuses autres branches des mathématiques. Le problème des mariages stables semble idéal dans ce contexte, car il ne fait appel à aucune connaissance préalable en Algorithmique, et conduit naturellement à illustrer les techniques essentielles de l'Analyse des

### Premier exemple

$$\mathcal{G} = \{g_1; g_2\} \quad \mathcal{F} = \{f_1; f_2\}$$

Préférences :

$$\begin{array}{ll} g_1 : f_1; f_2 & f_1 : g_1; g_2 \\ g_2 : f_1; f_2 & f_2 : g_1; g_2 \end{array}$$

Un seul couplage stable :  $(g_1, f_1)$   $(g_2, f_2)$

Certains sont contents, d'autres moins.



### Troisième exemple

Préférences :

$$\begin{array}{ll} g_1 : f_1; f_2 & f_1 : g_2; g_1 \\ g_2 : f_2; f_1 & f_2 : g_1; g_2 \end{array}$$

Un couplage stable :  $(g_1, f_1)$   $(g_2, f_2)$

Les garçons sont contents

Un autre stable :  $(g_1, f_2)$   $(g_2, f_1)$

Les filles sont contentes

Qui est content ?



### Deuxième exemple

Préférences :

$$\begin{array}{ll} g_1 : f_1; f_2 & f_1 : g_1; g_2 \\ g_2 : f_2; f_1 & f_2 : g_2; g_1 \end{array}$$

Un seul couplage stable :  $(g_1, f_1)$   $(g_2, f_2)$

Tout le monde est content.



### Premier essai : le laisser-faire

- ▶ Former des couples au hasard
- ▶ Choisir une instabilité, et recombinaison les couples

$$(g_1, f_1)(f_2, g_2) \longrightarrow (g_1, f_2)(f_1, g_2) \text{ si } g_1 <_f f_2 \wedge f_2 <_{g_2} f_1$$

relation de réécriture, non-déterministe.

aboutit-on à un couplage stable ?

- ▶ Il est possible de boucler à l'infini.
- ▶ On aboutit presque sûrement à un couplage stable, parfois en un temps exponentiel en  $n$  (résultat récent : 1995)



## Le marché n'a pas toujours raison

### Contre-exemple

$G = \{a; b; x\}$  et  $F = \{A; B; X\}$ .

A :	b; a; x	a :	A; B; X
B :	a; b; x	b :	B; A; X
X :	peu importe	x :	peu importe

$\{(b, X); (a, B); (x, A)\} \rightarrow \{(b, X); (x, B); (a, A)\} \rightarrow$   
 $\{(b, A); (x, B); (a, X)\} \rightarrow \{(b, B); (x, A); (a, X)\} \rightarrow$   
 $\{(b, X); (a, B); (x, A)\} \rightarrow \dots$

Cela dit, il existe des couplages stables.

$\{(x, X); (b, A); (a, B)\}$  ou  $\{(x, X); (a, A); (b, B)\}$ .

Comment traite-t-on le cas général ?



## L'Algorithme de Gale et Shapley

Un couplage qui évolue au cours de l'algorithme (les fiancés)

**tant que**  $\left\{ \begin{array}{l} \text{il existe un homme } h \text{ qui est libre et n'a pas} \\ \text{demandé toutes les femmes en mariage} \end{array} \right. \iff$  **non-déterminisme !!**

**soit**  $f$  la femme préférée de  $h$  parmi celles qu'il n'a pas encore demandées en mariage

si  $f$  est libre

**alors**  $h$  et  $f$  se fiancent

**sinon**

**soit**  $h'$  le fiancé de  $f$

si  $f$  préfère  $h$  à  $h'$

**alors**  $f$  quitte  $h'$  (qui redevient libre) et se fiance avec  $h$

Questions :

- ▶ Termine ? combien de temps ?
- ▶ sur un couplage ? parfait ? stable ?



## Terminaison

Prouver la terminaison, un problème classique un informatique...

Qu'est-ce qui décroît ? Trouver une *mesure* ou un *variant*

Chaque homme ne demande jamais deux fois la même femme en mariage.

Chaque "proposition"  $(h, f)$  n'est faite qu'au plus une fois.

Une proposition par tour de boucle

La boucle principale est effectuée au plus  $n^2$  fois

Complexité du programme :  $O(n^2)$

si le corps de la boucle est exécuté en temps constant.



## Correction (1) : couplage

Lorsque l'algorithme s'arrête, c'est sur un couplage (personne n'est fiancé deux fois en même temps).

C'est vrai au début (personne n'est fiancé)

C'est *préservé* à chaque tour de boucle :

- ▶ seuls des hommes libres se fiancent
- ▶ les femmes qui se fiancent sont soit libres, soit elles quittent leur précédent fiancé.

Formellement, on a fait une récurrence sur le nombre de tours de boucle

On a montré que la propriété était un *invariant*



### Correction (2) : couplage parfait

Est-ce que tout le monde est fiancé à la fin ?

On montre d'abord :

(a) Une femme qui a reçu une proposition reste fiancée jusqu'à la fin (invariant facile)

On s'arrête si :

- ▶ Tous les hommes sont fiancés *mais alors toutes les femmes le sont aussi*
- ▶ ou bien les hommes libres n'ont plus de avances à faire *mais alors ils ont fait des avances à toutes les femmes, donc toutes les femmes sont fiancées*

On a utilisé :

- 1) Un invariant supplémentaire (a)
- 2) la condition d'arrêt de la boucle.

### Correction (3) : Stabilité

Est-ce que le couplage est stable ?

On raffine la propriété :

(a) Une femme qui a reçu une proposition reste fiancée jusqu'à la fin et ses fiancés successifs lui plaisent de plus en plus

Soient deux couples du résultat :  $(h, f)$  et  $(h', f')$

Supposons :

$$f' \succ_h f \wedge h \succ_{f'} h'$$

On sait alors que  $h$  a fait une proposition à  $f$ . Or, puisque  $h$  préfère  $f'$  à  $f$ , il a, précédemment, au cours de l'exécution, fait une proposition à  $f'$ .

Si  $f'$  a accepté cette proposition, son fiancé actuel  $h'$  doit lui plaire plus que  $h$  (d'après (a)).

Si  $f'$  n'a pas accepté cette proposition, son fiancé du moment lui plaisait plus. D'après (a) son fiancé final  $h'$  lui plait (encore) plus.

Le couplage est stable.

Remarque : on n'a pas utilisé la propriété d'arrêt. La stabilité (partielle) est un invariant.

## Que dire de plus ?

- ▶ Non-déterminisme
- ▶ Quel couplage quand il y a plusieurs solutions

### Un algorithme injuste

L'algorithme favorise les garçons !

**Définition** : on dit que  $f$  (resp.  $g$ ) est un(e) partenaire valide de  $g$  (resp.  $f$ ) s'il existe un couplage stable contenant  $(g, f)$ .

**best( $h$ ) est la meilleure partenaire valide de  $h$  (du point de vue de  $h$ )**

**worse( $f$ ) le pire partenaire valide de  $f$  (du point de vue de  $f$ ).**

$$S^* = \{(h, \text{best}(h)) \mid h \in H\}$$

**Théorème** Toute exécution de l'algorithme de Gale-Shapley termine en produisant l'ensemble  $S^*$ .

## Un algorithme injuste

**Théorème** Toute exécution de l'algorithme de Gale-Shapley termine en produisant l'ensemble  $S^*$ .

Surprenant :

- ▶ pas évident à première vue que  $S^*$  est un couplage stable
- ▶ montre en plus que Gale-Shapley est en fait déterministe

Par ailleurs, bon exemple de preuve plus difficile : l'invariant est plus fort que la propriété finale.

au cours de l'exécution :

1. si  $(h, f)$ , alors  $f \geq_h \text{best}(h)$
2. si  $h$  pas fiancé, il n'a fait des avances qu'à des  $f >_h \text{best}(h)$

Pour être clair, soyons précis : récurrence sur le nombre de tours de boucle effectués. On montre 1 et 2 ensemble.

**0 tours** : OK (pas d'avances, pas de fiancés)



1. si  $(h, f)$ , alors  $f \geq_h \text{best}(h)$
2. si  $h$  pas fiancé, il n'a fait des avances qu'à des  $f >_h \text{best}(h)$

**$n$  tours  $\implies n + 1$  tours**

$h$  fait des avances à  $f$ .

1. D'après HR2,  $h$  n'a fait des avances qu'à des  $f' >_h \text{best}(h)$ . Comme  $f$  est la suivante,  $f \geq_h \text{best}(h)$ .

2.  $f$  était fiancée à  $h'$ . Deux cas :

- ▶ Si  $f$  ne se fiance pas à  $h$  et préfère  $h'$ , il faut montrer  $f >_h \text{best}(h)$ .

Il suffit que  $f \neq_h \text{best}(h)$ . Soit un couplage stable avec  $(h, f)$ .

$(h', f')$  avec  $f' \leq_{h'} \text{best}(h') \leq_{h'} f$  (par HR1). Donc  $f' <_{h'} f$ .

- ▶ Si  $f$  préfère  $h$  à  $h'$ . M.q.  $f >_{h'} \text{best}(h')$  cad.  $f \neq \text{best}(h')$ .

Si  $(h', f)$  dans un couplage stable, dans ce couplage  $(h, f')$  avec  $f' \leq_h \text{best}(h) \leq_h f$ . Comme  $f \neq f'$ , on a aussi  $f' <_h f$ .



## Conclusions supplémentaires

- ▶ L'algorithme de Gale-Shapley est en fait déterministe.

**Corollaire** Toutes les exécutions de Gale-Shapley rendent le même résultat.

- ▶ L'algorithme est très favorable à l'une des deux parties (ici les garçons).

Plus généralement, sur les preuves par invariant :

- ▶ Ce sont des preuves par récurrence ; sur les cas compliqués, il est utile d'expliciter cela.
- ▶ Dans des cas plus compliqués, la difficulté est de formuler le bon invariant.



## Un algorithme très injuste !

L'algorithme de Gale-Shapley rend l'ensemble suivant :

$$\{\text{worse}(f), f\} \mid f \in F\}.$$

Preuve : en exercice.

Il n'est pas nécessaire de refaire une récurrence ou de trouver un invariant.

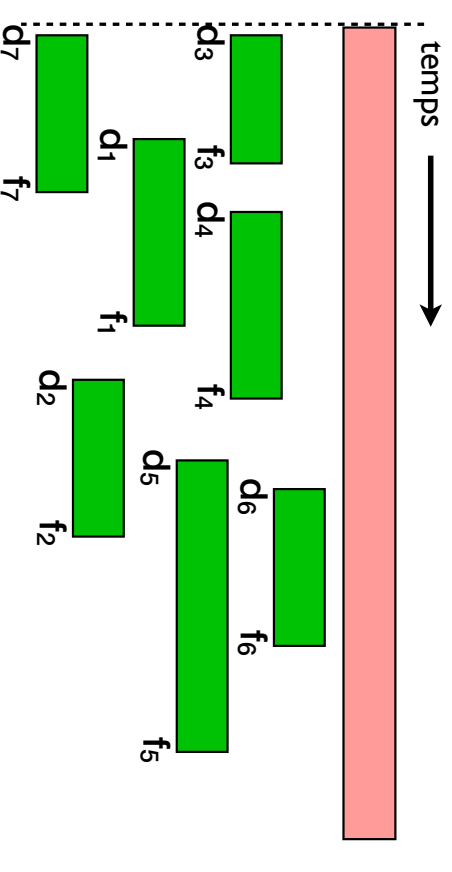




# Ordonnancement d'intervalles

Idée de l'algorithme

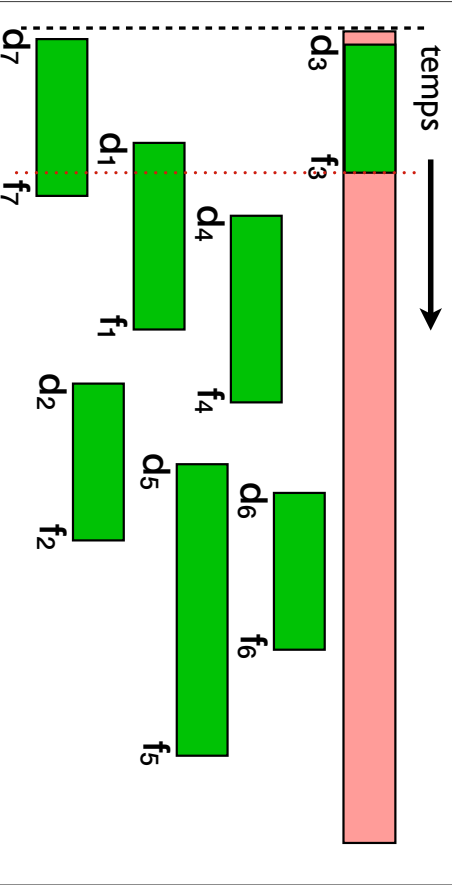
Des demandes d'utilisation  $(d_1, f_1) \dots (d_n, f_n)$



# Ordonnancement d'intervalles

Idée de l'algorithme

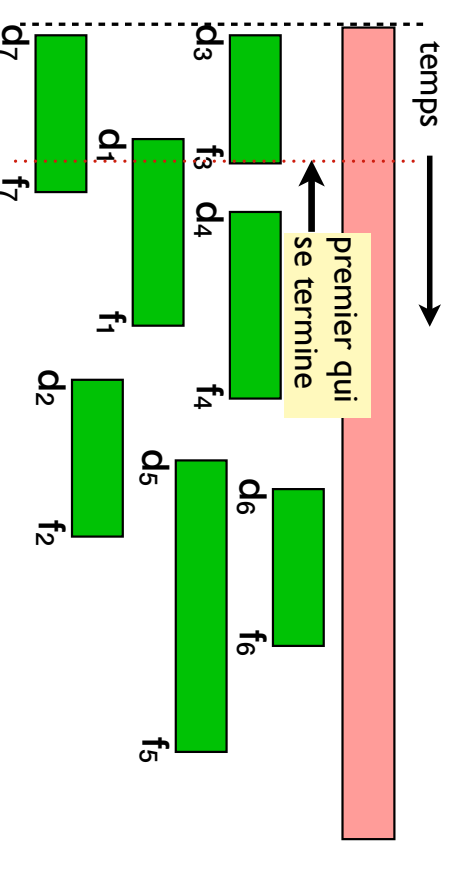
Des demandes d'utilisation  $(d_1, f_1) \dots (d_n, f_n)$



# Ordonnancement d'intervalles

Idée de l'algorithme

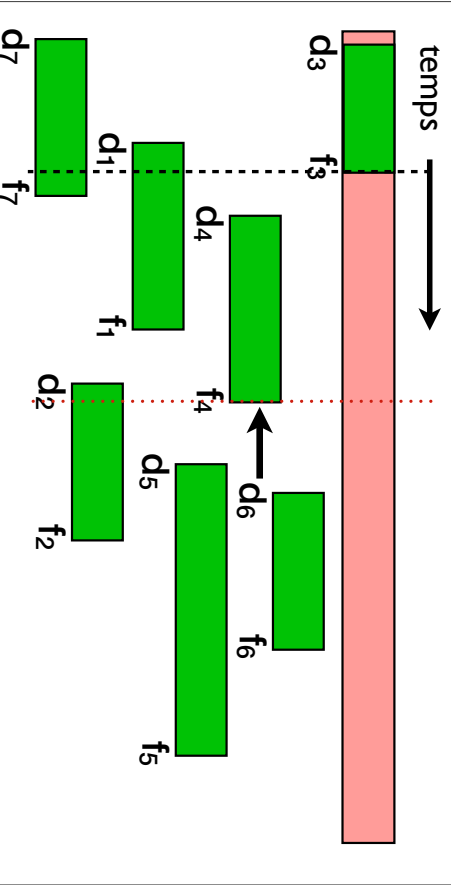
Des demandes d'utilisation  $(d_1, f_1) \dots (d_n, f_n)$



# Ordonnancement d'intervalles

Idée de l'algorithme

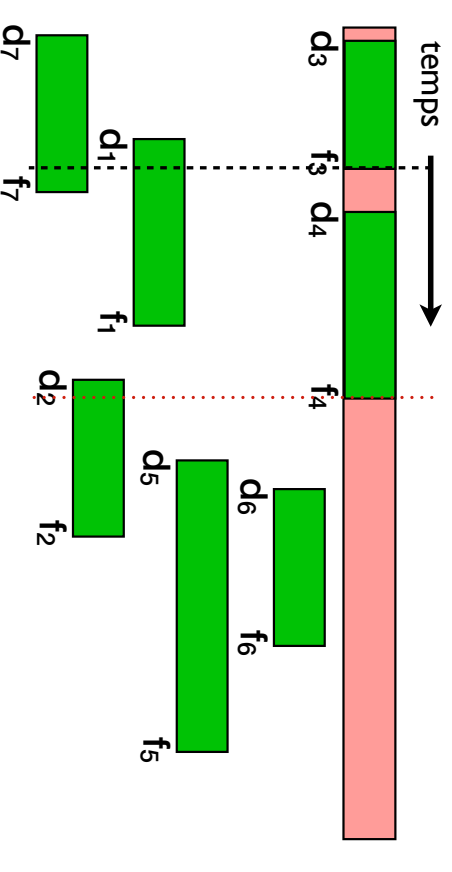
Des demandes d'utilisation  $(d_1, f_1) \dots (d_n, f_n)$



# Ordonnancement d'intervalles

Idée de l'algorithme

Des demandes d'utilisation  $(d_1, f_1) \dots (d_n, f_n)$

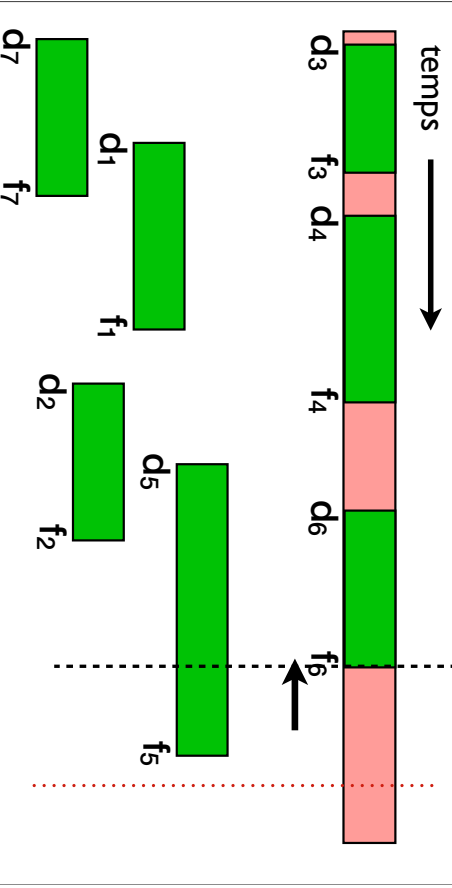


mardi 31 janvier 2012

# Ordonnancement d'intervalles

Idée de l'algorithme

Des demandes d'utilisation  $(d_1, f_1) \dots (d_n, f_n)$

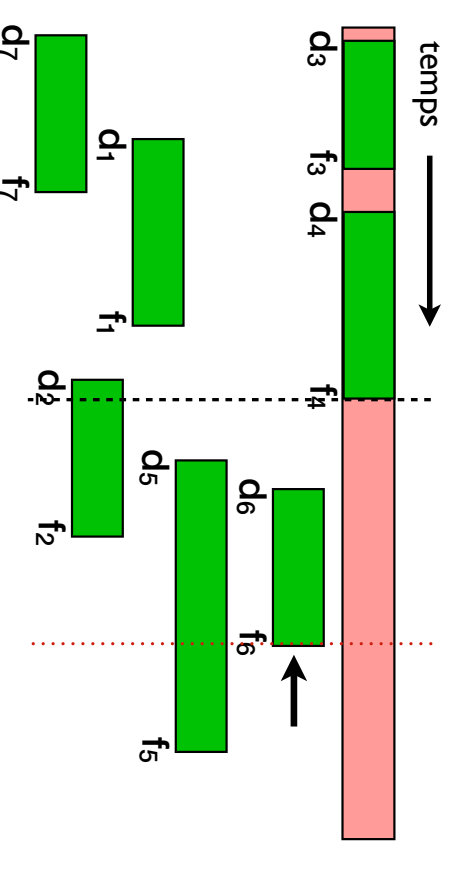


mardi 31 janvier 2012

# Ordonnancement d'intervalles

Idée de l'algorithme

Des demandes d'utilisation  $(d_1, f_1) \dots (d_n, f_n)$

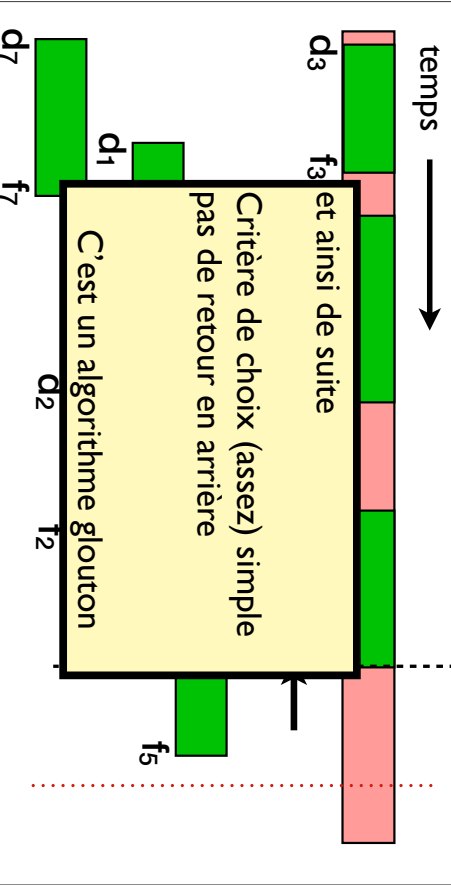


mardi 31 janvier 2012

# Ordonnancement d'intervalles

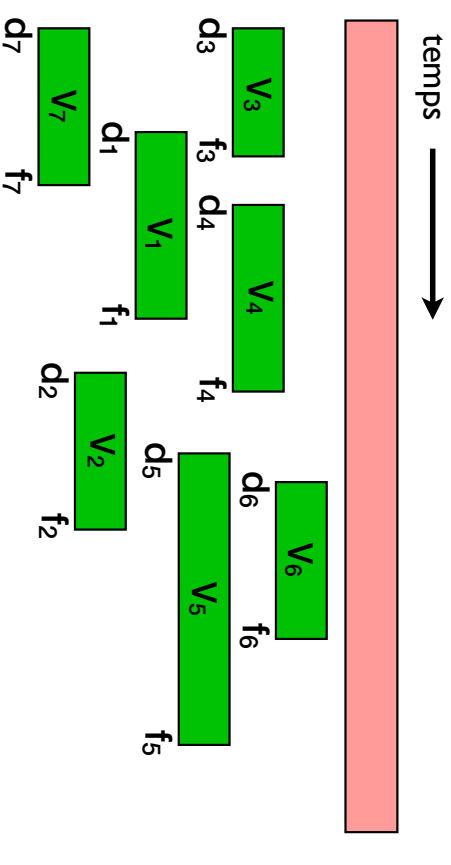
Idée de l'algorithme

Des demandes d'utilisation  $(d_1, f_1) \dots (d_n, f_n)$



mardi 31 janvier 2012

# Ordonnancement d'intervalles valués

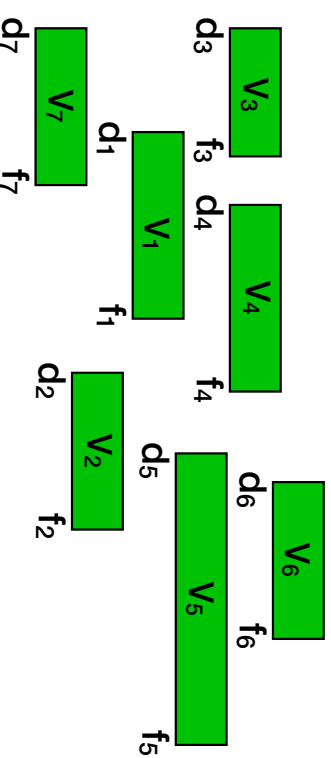


mardi 31 janvier 2012

# Ordonnancement d'intervalles

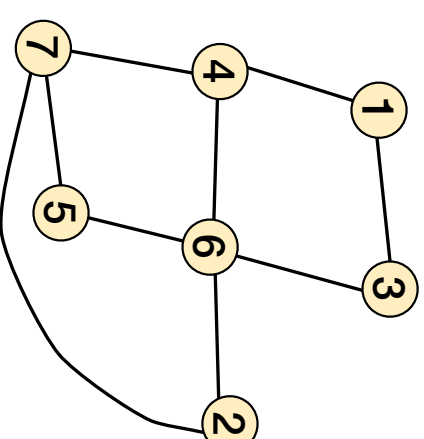
Pareil mais une valeur sur chaque demande  
maximiser la somme des valeurs satisfaites

te Algorithme très différent: programmation dynamique  
(utilisation de beaucoup de mémoire)



mardi 31 janvier 2012

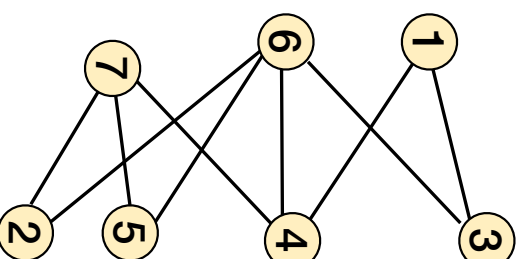
# Graphes bi-parties



mardi 31 janvier 2012

# Graphes bi-parties

- Déterminer si un graphe est bi-partie: parcours de graphe en largeur d'abord
- Extraire un couplage à partir d'un graphe bi-partie: optimisation de flux



mardi 31 janvier 2012

# Conclusion

- Cycle: problème, description, terminaison, correction
- Invariant de boucle
- Diversité des techniques algorithmiques

## La suite

Cet après-midi: exercices d'algorithmique

**Pour le 15 février: implémentation de Gale-Shapley  
(devoir à la maison, 2 semaines)**

**Semaine prochaine: Programmation orienté-objet**