INTRODUCTION À L'ALGORITHMIQUE

-

LES ALGORITHMES GLOUTONS

Chargée de cours: Lélia Blin
Transparents:http://www-npa.lip6.fr/~blin/Enseignements.html
Email: lelia.blin@lip6.fr

CONCEPTION D'ALGORITHMES EFFICACES

Il existe des techniques pour la conception d'algorithme efficace comme

- Diviser pour régner
- Résolution par récurrence
- Algorithme glouton
- Programmation dynamique

LES ALGORITHMES GLOUTONS

Plan du cours :

- Le choix d'activités
- Les pièces de monnaie
- Le problème du sac à dos
- Codage de Huffman

ALGORITHME GLOUTON

Un algorithme glouton

fait toujours le choix

qui semble le meilleur

sur le moment

ALGORITHME GLOUTON

Un algorithme glouton

fait un choix localement optimal

dans l'espoir que ce choix mènera

à la solution globalement optimale

ALGORITHME GLOUTON

les algorithmes gloutons

ne conduisent pas toujours à la solution optimale

LE CHOIX D'ACTIVITÉS

Lélia Blin

Université d'Evry

LE PROBLÈME DU CHOIX D'ACTIVITÉS

Soit n activités concurrentes

noté par un ensemble S={1,2,…..,n}

qui souhaitent utiliser une ressource (ex: salle de cours)

qui ne peut être utilisée que pour une activité à la fois.

ACTIVITÉS

Chaque activité i possède:

un horaire de début

noté: d_i

un horaire de fin

noté f_i

avec $d_i \leq f_i$

COMPATIBILITE

Une activité i à lieu pendant l'intervalle de temps $[d_i, f_i[$

Les activités i et j sont compatibles si

les intervalles $[d_i, f_i[$ et $[d_j, f_j[$

ne se super-posent pas (si $d_i \geq f_j$ ou $d_j \geq f_i$)

LE PROBLÈME DU CHOIX D'ACTIVITÉS: PCA

Le problème du choix d'activités est de

choisir le plus grand nombre

possible d'activités compatibles entre elles.

ORDONNANCEMENTS

Ordonnancements d'intervalles sur une machine

Définition d'intervalles

Soit $E=\{\sigma_1, ..., \sigma_n\}$ un ensemble d'intervalles

tels que pour tout i, on définit

$\sigma_i=[a,b[$ avec a et b deux réels tel que $a<b$

DÉFINITION D'ORDONNANCEMENT D'INTERVALLES

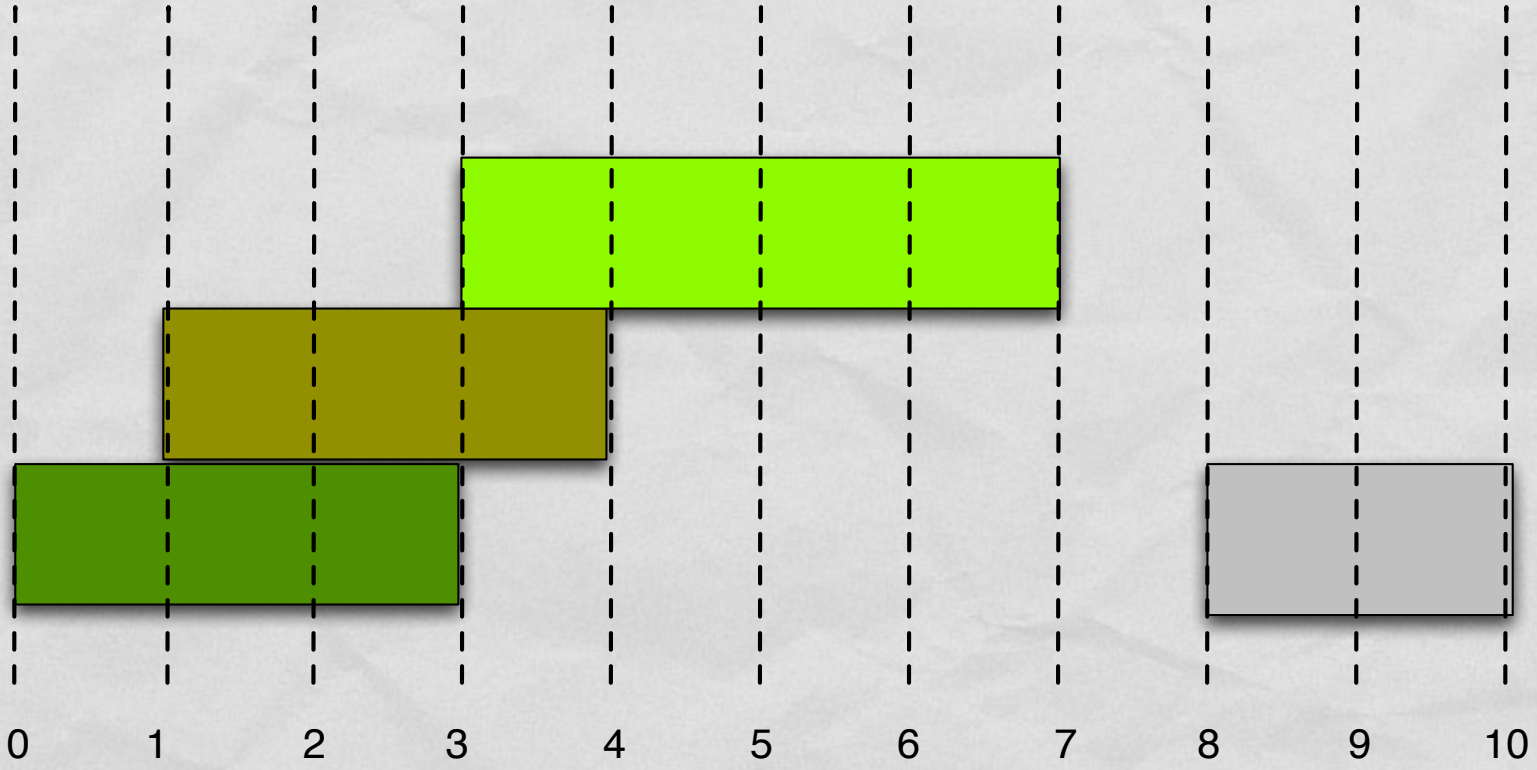
θ est un ordonnancement de $E=\{\sigma_1, \ldots, \sigma_n\}$

si et seulement si

$\theta \subseteq E$ et $\forall\ \sigma_a, \sigma_b$ avec $\sigma_a \neq \sigma_b$ on a $\sigma_a \cap \sigma_b = \varnothing$

EXEMPLE

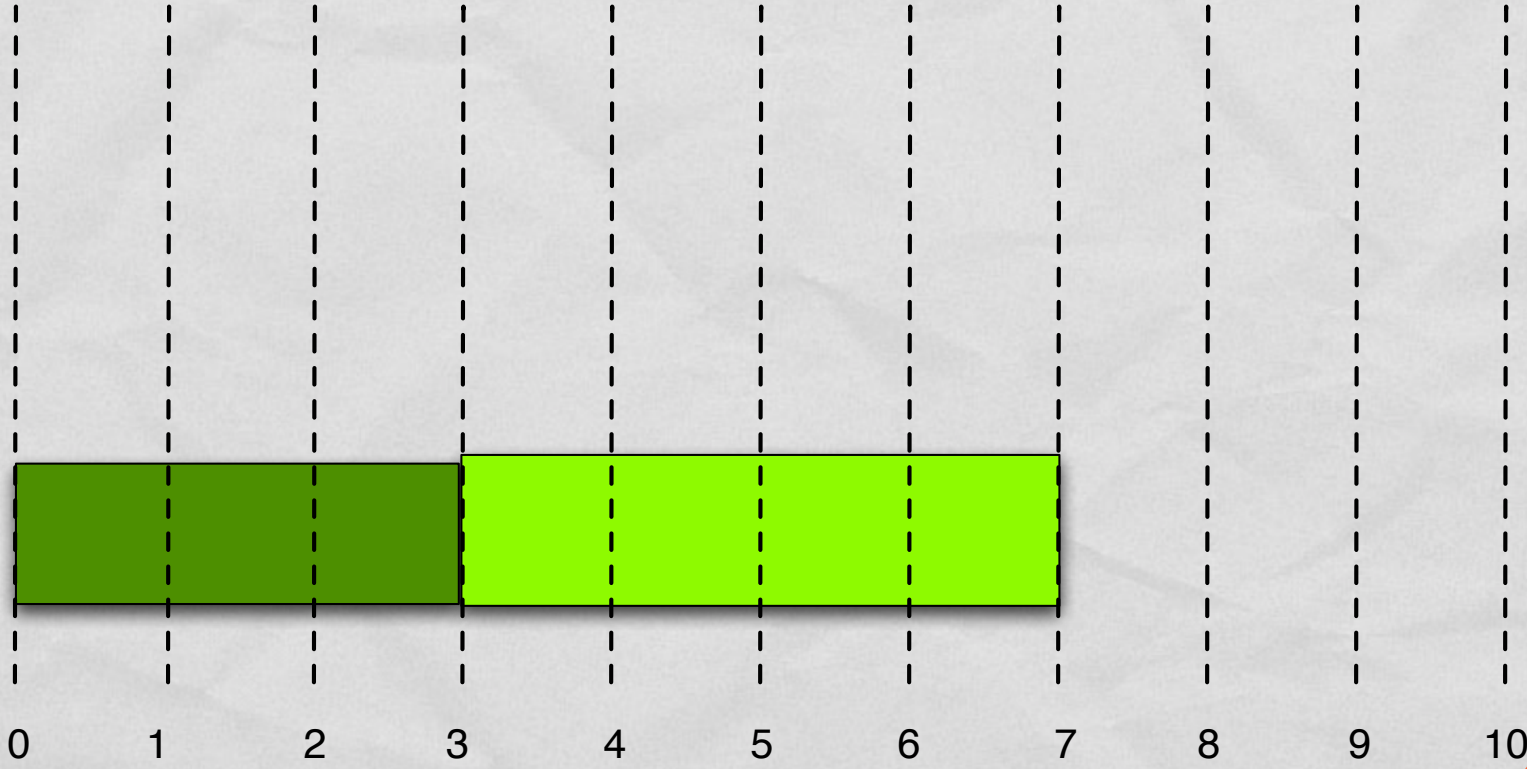
E={[0,3[,[1,4[,[3,7[,[8,10[}



EXEMPLE

E={[0,3[,[1,4[,[3,7[,[8,10[}

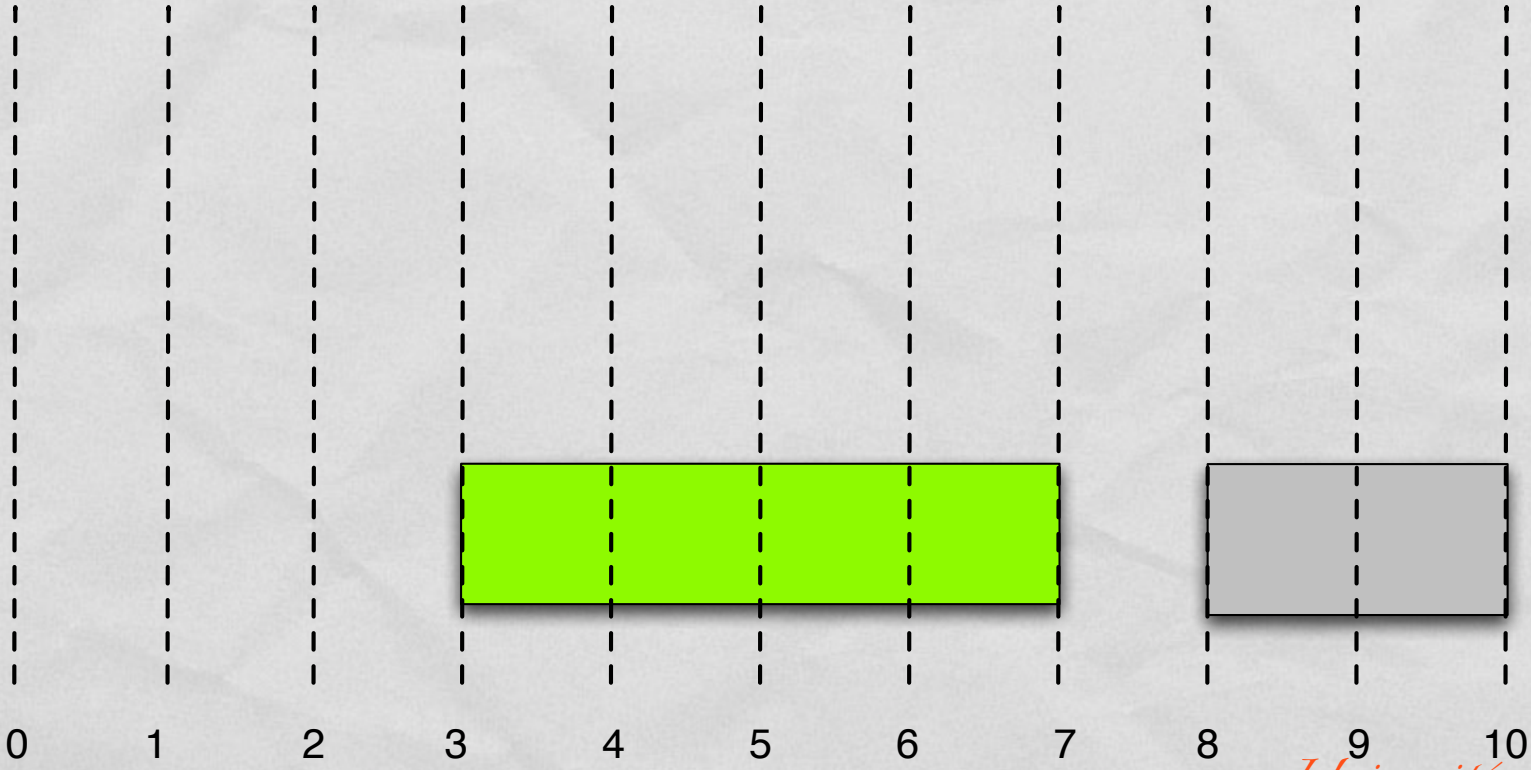
θ_1= {[0,3[,[3,7[} est un ordonnancement de E



EXEMPLE

E={[0,3[,[1,4[,[3,7[,[8,10[}

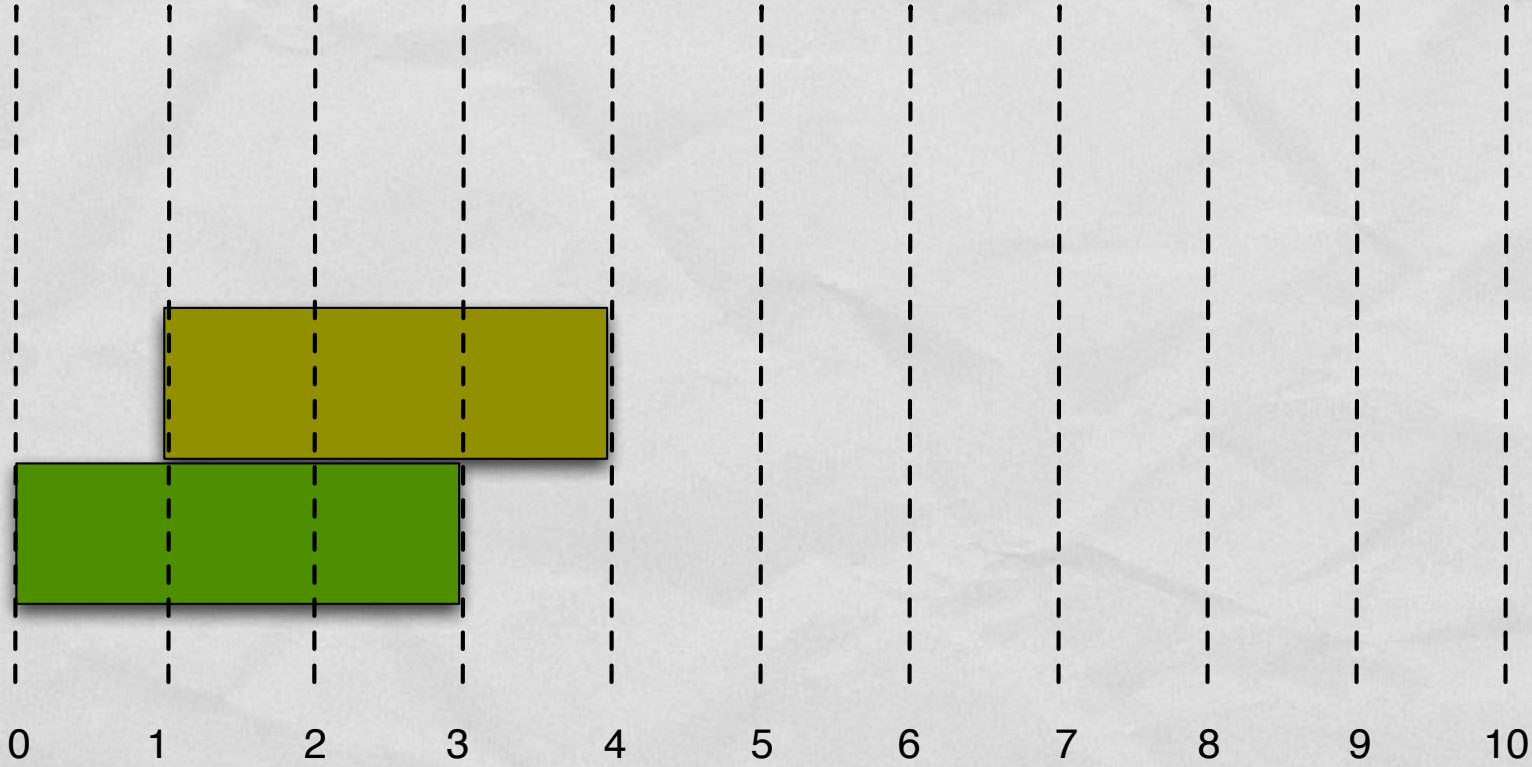
θ_2= {[3,7[,[8,10[} est un autre ordonnancement de E



EXEMPLE

E={[0,3[,[1,4[,[3,7[,[8,10[}

θ_3= {[0,3[,[1,4[,} n'est pas un ordonnancement de E



REMARQUE

On note que les intervalles sont ouverts à droite afin d'avoir:

$\forall a<b<c$ on ait $[a,b[\cap[b,c[=\emptyset$

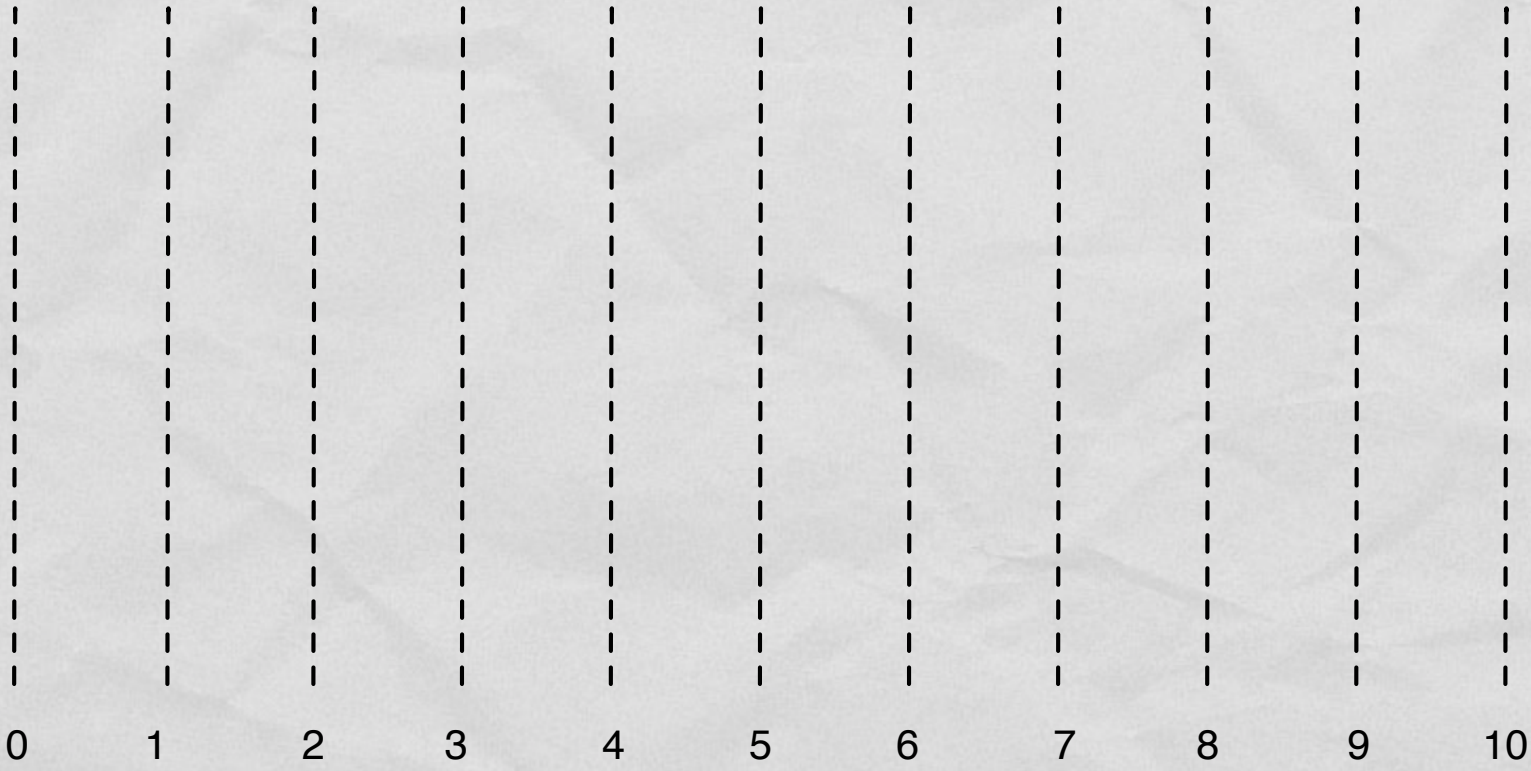
sinon $[a,b]\cap[b,c]\neq\emptyset$

ALGORITHME D'ORDONNANCEMENT QUELCONQUE

```
Début
θ=∅
Pour tout σ∈E
        Faire si (pour tout σ'∈E, on a σ∩σ'=∅)
                alors θ←θ∪{σ}
         Fin si
Retourner θ
FIN
```

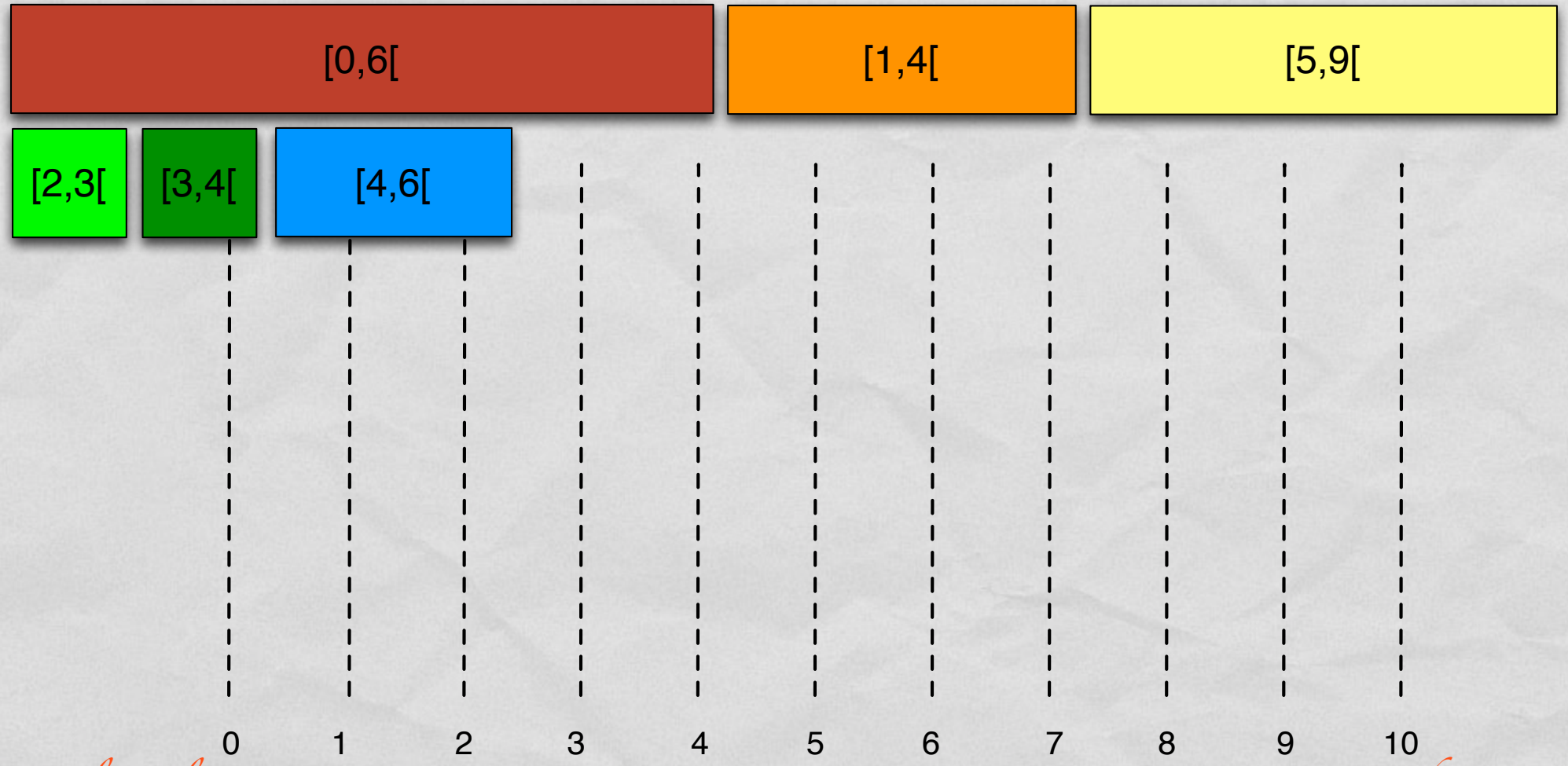
EXEMPLE D'EXÉCUTION

E={[0,6[,[1,4[,[5,9[,[2,3[,[3,4[,[4,6[}



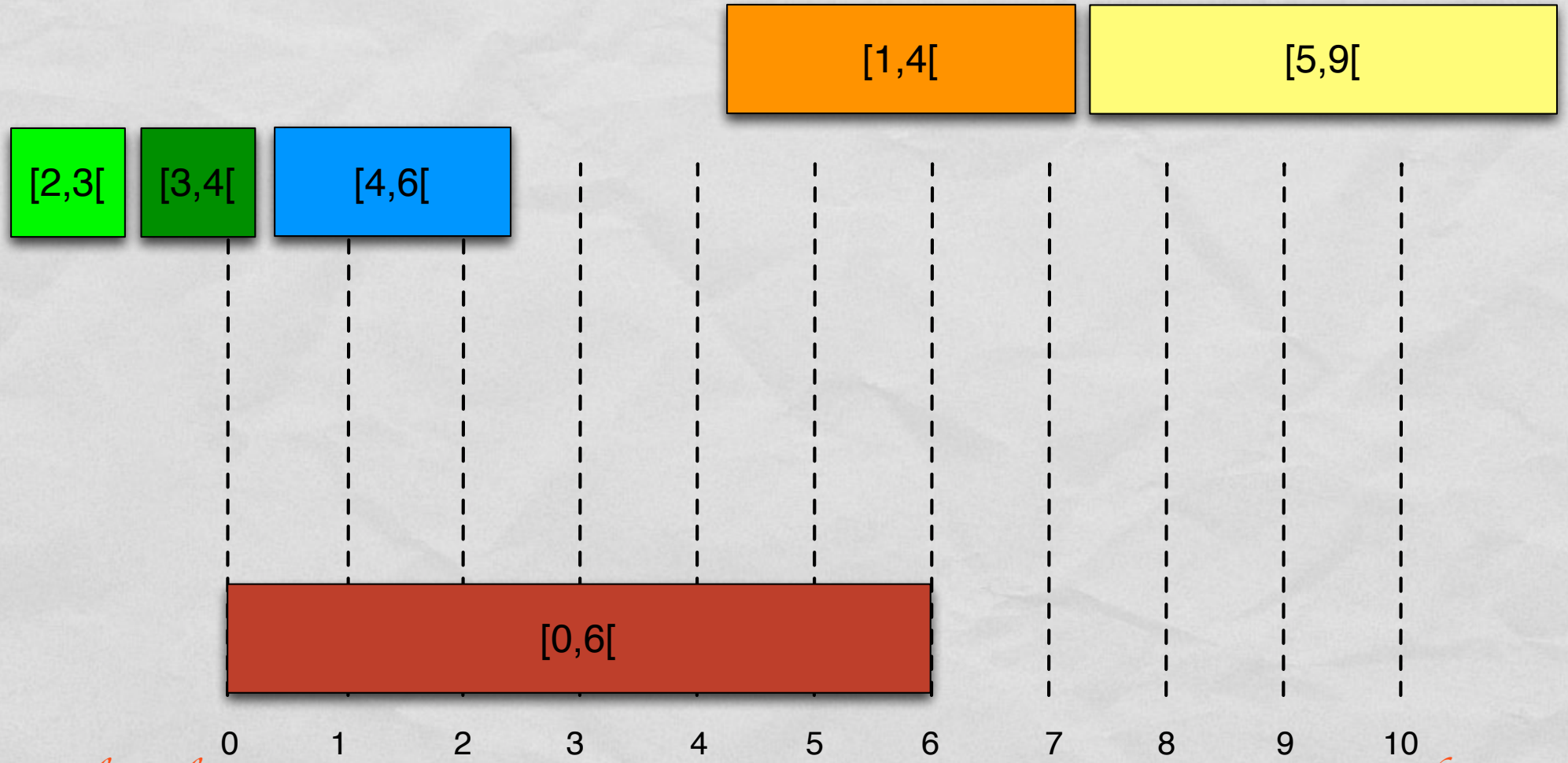
EXEMPLE D'EXÉCUTION

E={[0,6[,[1,4[,[5,9[,[2,3[,[3,4[,[4,6[}



EXEMPLE D'EXÉCUTION

E={[0,6[,[1,4[,[5,9[,[2,3[,[3,4[,[4,6[}



ORDONNANCEMENT QUELCONQUE

Si nous reprenons le problème

De maximiser le nombre de cours

Dans une salle de cours

Nous avons vu un exemple

qui prouve que la solution proposée

ne retourne pas la solution optimale

INTERVALLES DE DURÉE MINIMUM

Intuitivement

Favorisé les cours de petite durée

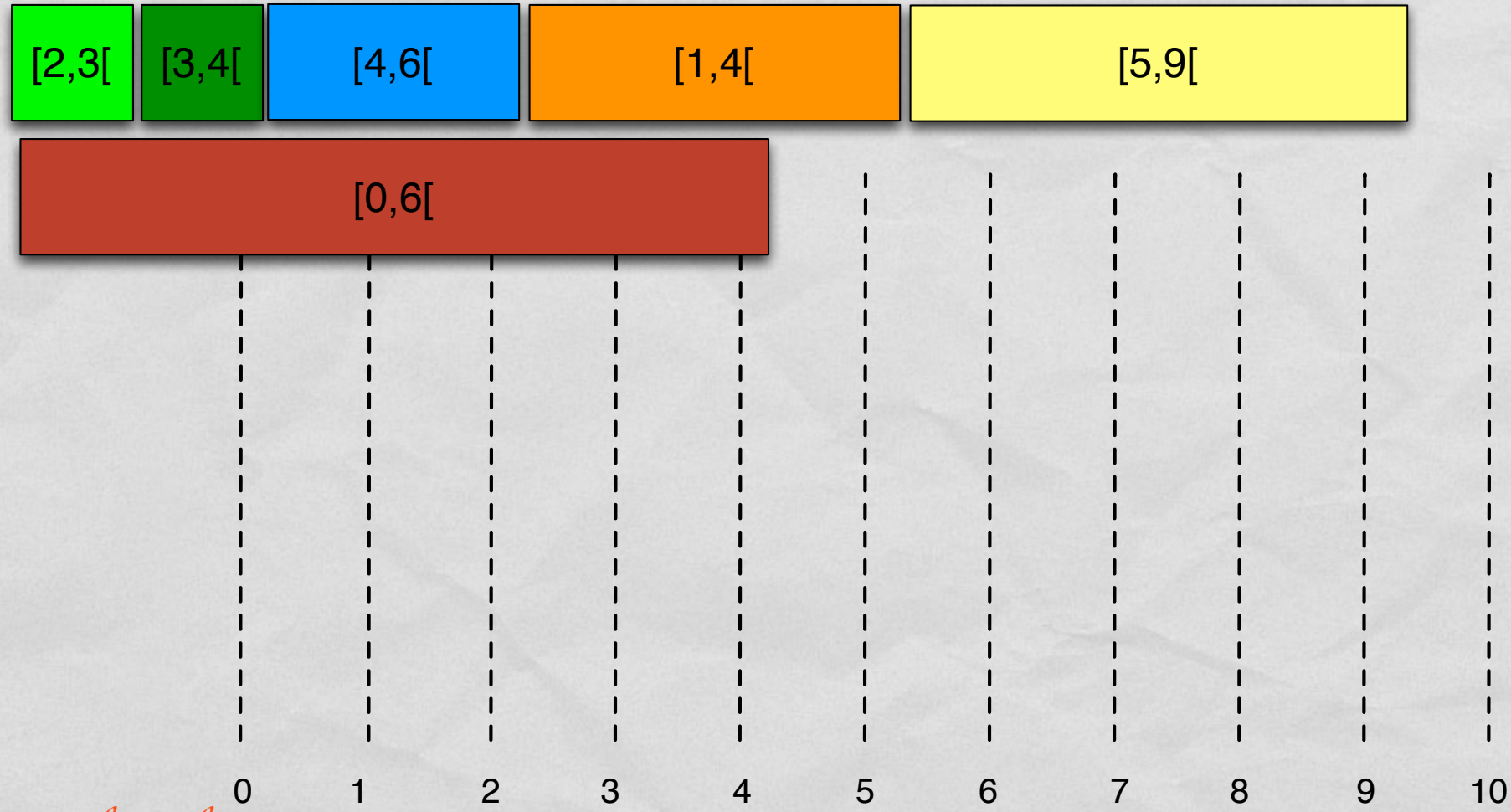
Ordonnée de façon croissante les cours suivants leurs durées

ALGORITHME PETITE TAILLE AVANT

```
Début
θ=∅
Pour tout σ∈E (trié dans l'ordre croissant de leurs durée
l=b-a)
        Faire si (pour tout σ'∈E, on a σ∩σ'=∅)
                alors θ←θ∪{σ}
         Fin si
Retourner θ
FIN
```

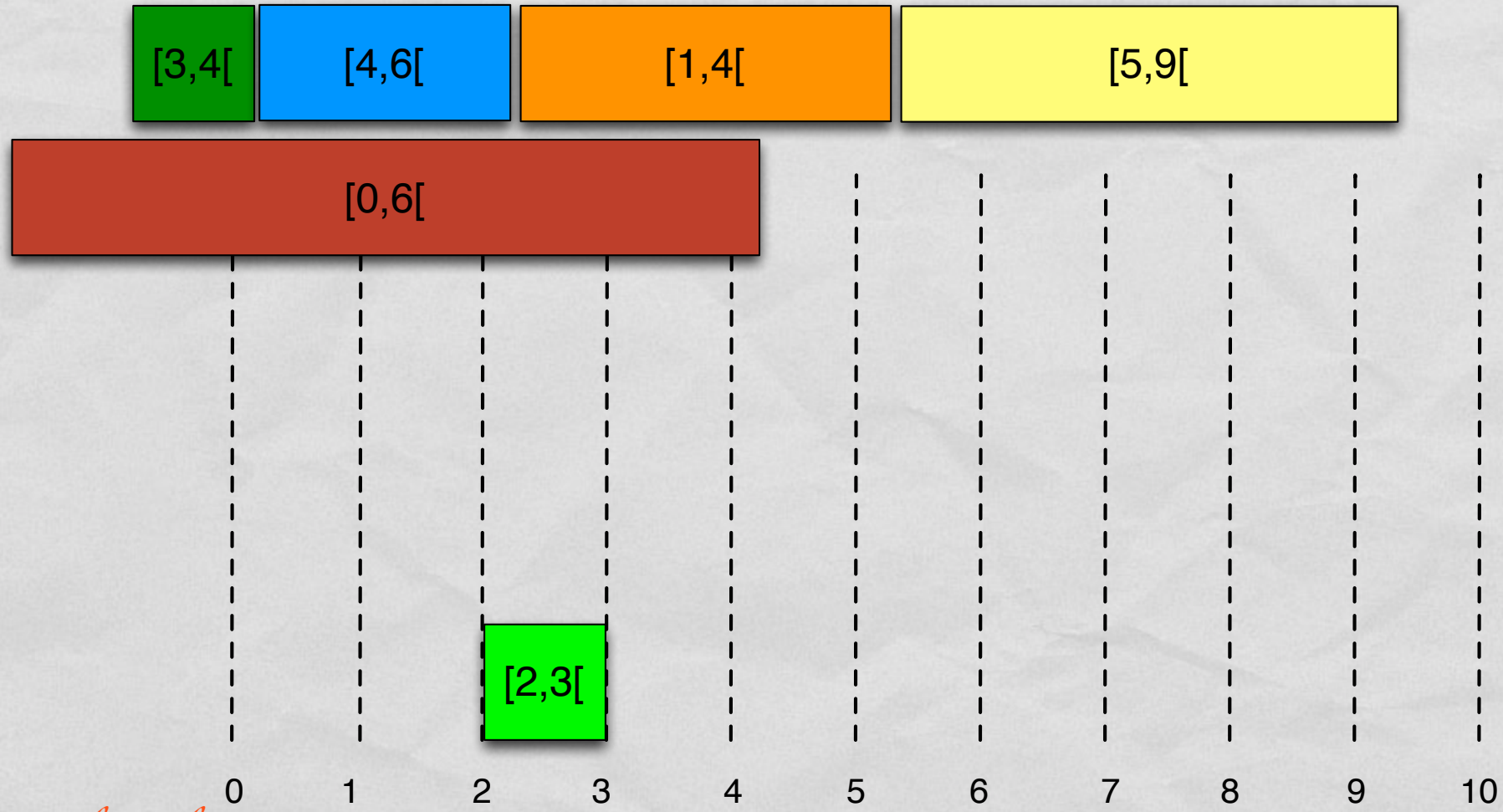
EXEMPLE D'EXÉCUTION

E={[0,6[,[1,4[,[5,9[,[2,3[,[3,4[,[4,6[}



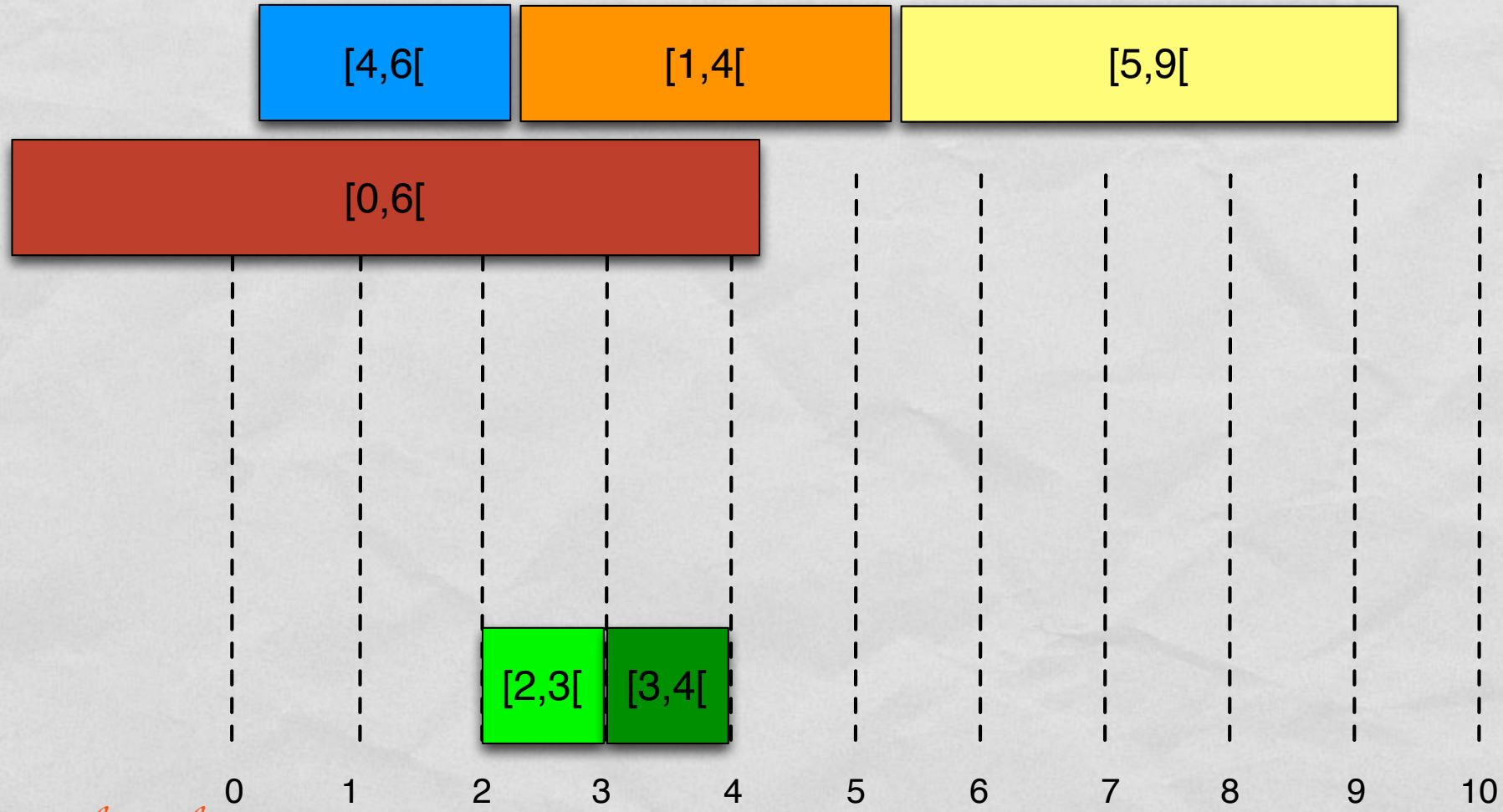
EXEMPLE D'EXÉCUTION

E={[0,6[,[1,4[,[5,9[,[2,3[,[3,4[,[4,6[}



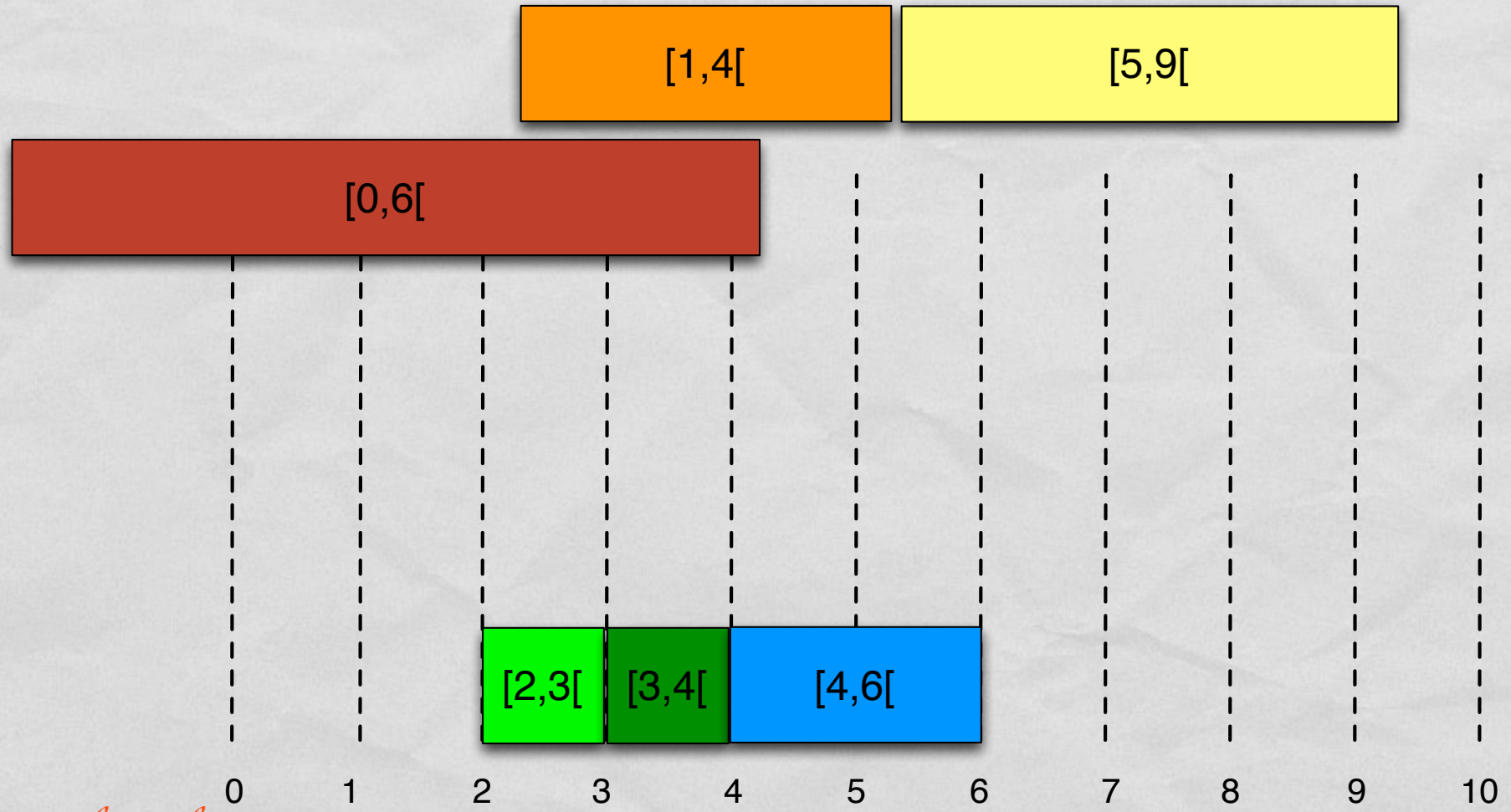
EXEMPLE D'EXÉCUTION

E={[0,6[,[1,4[,[5,9[,[2,3[,[3,4[,[4,6[}



EXEMPLE D'EXÉCUTION

E={[0,6[,[1,4[,[5,9[,[2,3[,[3,4[,[4,6[}



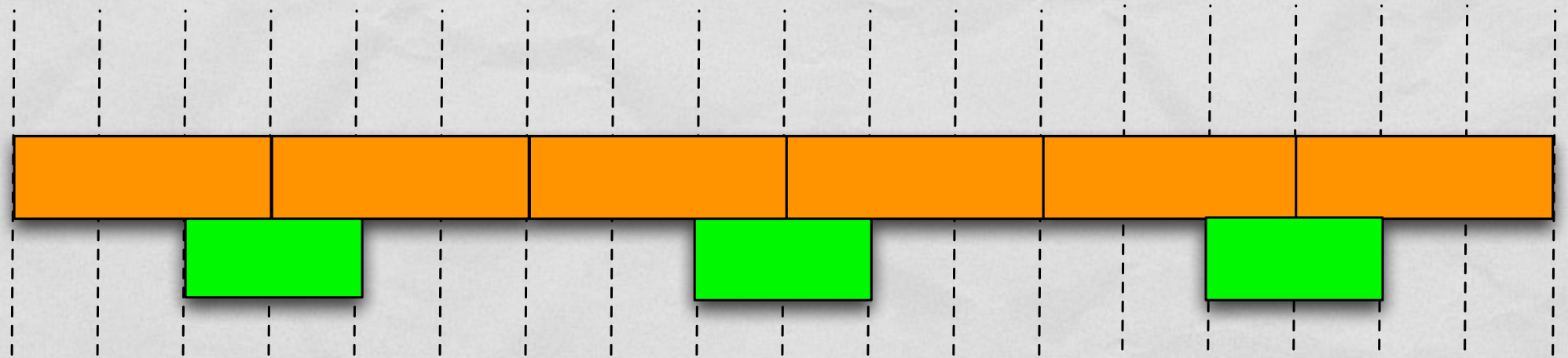
ALGORITHME PETITE TAILLE AVANT

Question:

Est-ce que cet algorithme garantit toujours qu'un maximum d'intervalles sera ordonnancés?

ALGORITHME PETITE TAILLE AVANT

Contre exemple:



ORDONNANCEMENT OPTIMAL

Si on définit θ^* comme l'ordonnancement optimal

On a alors dans notre exemple

$|\theta^*|=2 > |\theta|=1$

ALGORITHME PETITE TAILLE AVANT: PTA

Question:

Est-ce que l'algorithme petite taille avant nous garantit quelque chose sur le nombre d'intervalles qui sera ordonnancés?

Réponses:

Oui, on sait que dans le pire des cas on a la propriété suivante:

Soit θ l'ordonnancement de E renvoyé par l'algo

Soit θ^* l'ordonnancement optimal de E

On a $\forall E, 2|\theta| \geq |\theta^*|$

ALGORITHME PETITE TAILLE AVANT

Réponse suite:

Cela signifie que l'on est sûr d'avoir

- Une solution relativement proche
- De la meilleure solution possible
- Puisqu'il y a au pire un facteur 2 d'écart entre les 2 solutions

THEORÈME

Pour tous ensemble E d'intervalles, l'algorithme PTA construit un ordonnancement θ de E tel que:

$2|\theta| \geq |\theta^*|$ et où θ^* est l'ordonnancement optimal de E.

PREUVE

Montrons d'abord la proposition suivante :

Tout intervalle de θ^* intersecte au moins un intervalle de θ (1)

Supposons par l'absurde qu'il existe un intervalle $\sigma^* \in \theta^*$ qui n'intersecte aucun intervalle de θ.

Par définition, l'algorithme PTA examine tous les intervalles de E

donc y compris σ^*.

PREUVE

Or par définition, l'algorithme ordonnance l'intervalle qu'il est en train de traiter si celui-ci n'intersecte aucun intervalle de θ.

C'est le cas pour σ^*, donc σ^* est nécessairement ordonnancé dans θ, donc il existe un intervalle (ici σ^*) de θ^* qui intersecte $\sigma^* \in \theta \rightarrow$ Contradiction !

La proposition (1) est donc prouvée

PREUVE SUITE

On va maintenant montrer la proposition suivante :

Tout intervalle de θ intersecte au plus 2 intervalles de θ* (2)

Supposons par l'absurde qu'il existe au moins trois intervalles :

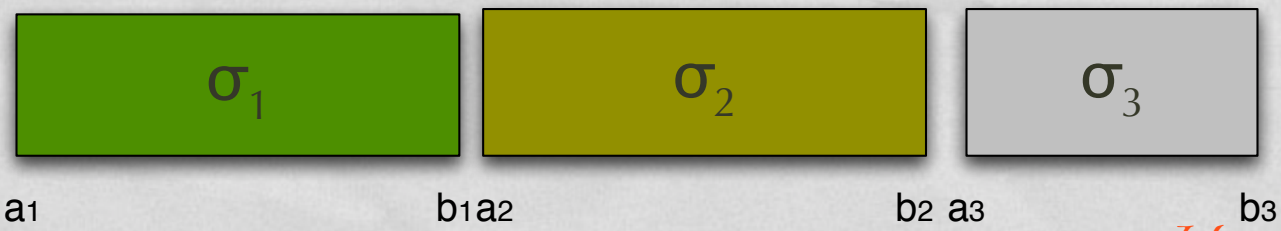
$\sigma_1 = [a_1,b_1[$ $\sigma_2 = [a_2,b_2[$ et $\sigma_3 = [a_3,b_3[$

Tels que σ_1,σ_2 et σ_3 intersectent $\sigma = [a,b[\in \theta$

Comme θ* est un ordonnancement $\sigma_1,\sigma_2,\sigma_3$ sont disjoints. Supposons alors, sans perte de généralité que l'on a :

$a_1 < b_1 \leq a_2 < b_2 \leq a_3 < b_3$

C'est-à-dire que $\sigma_1,\sigma_2,\sigma_3$ sont ordonnancés dans θ* dans l'ordre suivant :

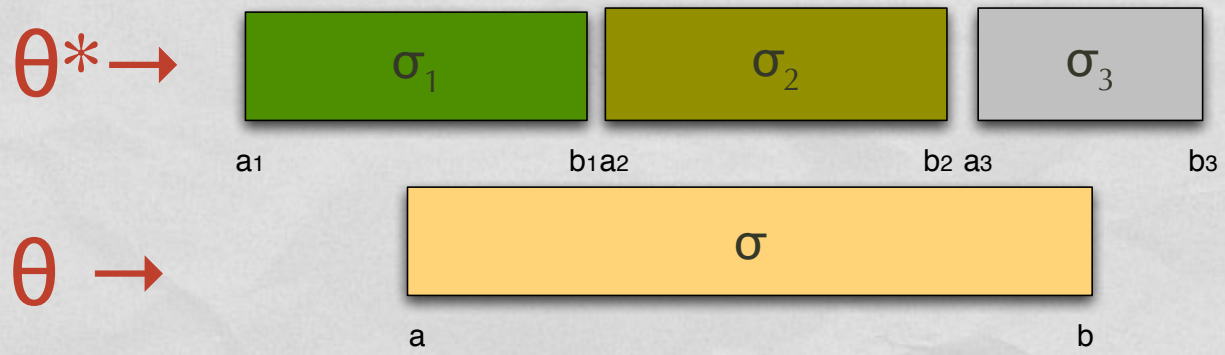


PREUVE SUITE

Comme $\sigma = [a,b[$ intersecte $\sigma_1, \sigma_2, \sigma_3$ on a nécessairement la relation suivante :

$$a_1 < b_1 \leq a_2 < b_2 \leq a_3 < b_3$$

Ce qui correspond à la relation suivante :



PREUVE SUITE

Comme $\sigma = [a,b[$ intersecte $\sigma_1, \sigma_2, \sigma_3$ on a:

$a < a_2 < b_2 < b$

et $\sigma_2 \in \theta^*$ plus petit que $\sigma \in \theta$

ce qui contredit la définition de l'algorithme PTA

car les intervalles sont traités par ordre croissant de leur longueur,

on aurait donc dû avoir σ_2 ordonnancé par l'algorithme à la place de σ.

La proposition (2) est donc prouvée.

PREUVE FIN

Soit θ' l'ensemble des intervalles de θ* qui intersectent un intervalle (ou plus) dans θ.

D'après la proposition (2) on a :

$|\theta'| \leq 2|\theta|$

Et d'après la proposition (1) on a $\theta' = \theta^*$ on en déduit :

$2|\theta| \geq |\theta^*|$

ALGORITHME GLOUTON: PSEUDO-CODE

On suppose les activités triées par ordre croissant de fin des horaires

$f_1 \leq f_2 \leq \ldots \leq f_n$

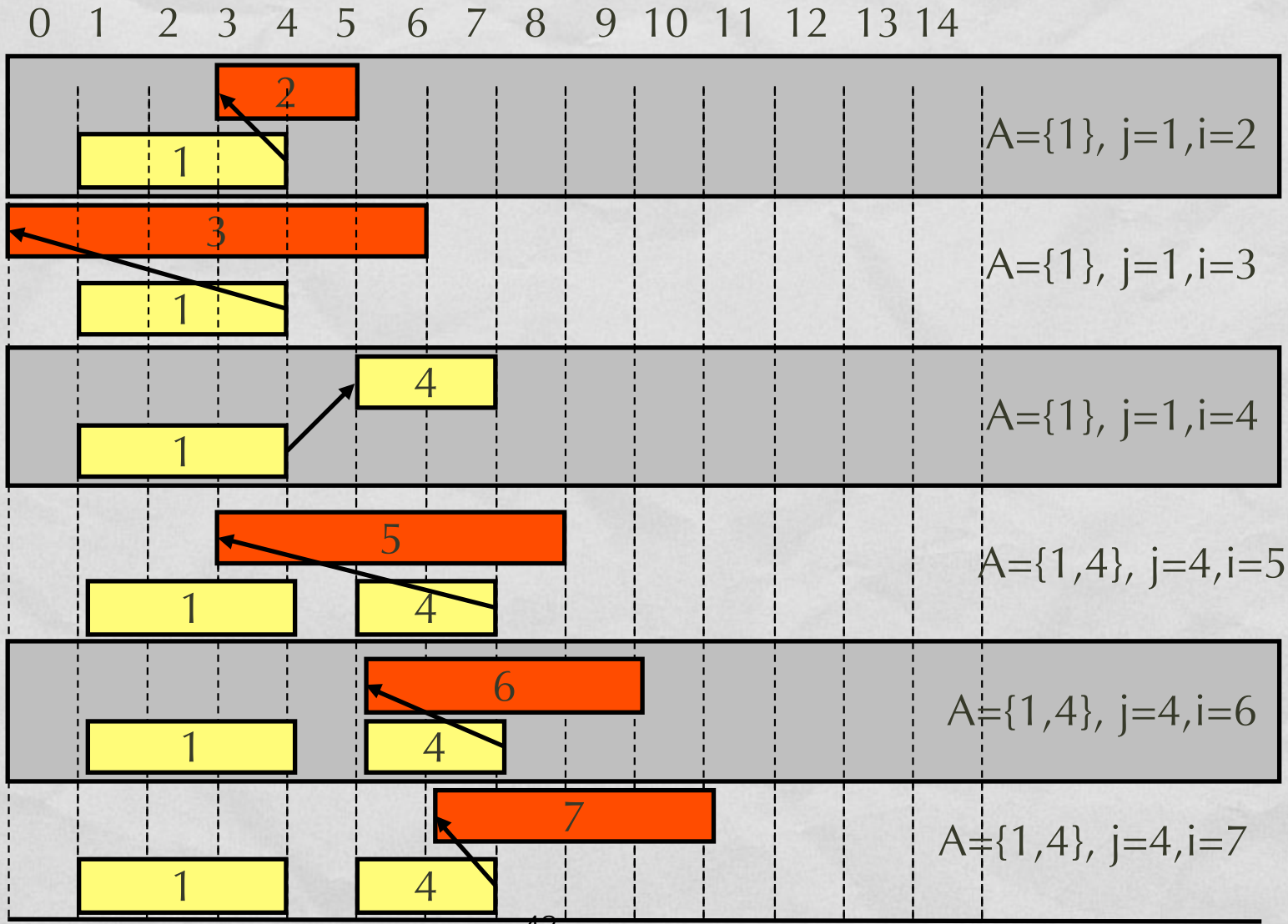
```
CHOIX-ACTIVITE-GLOUTON(s,f)
n ← longueur[s]
A ← {1}
j ← 1
pour i← 2 à n faire
   si di≥fj alors
      A ← AU{i}
      j ← i
retourner A
```

A collecte les activités sélectionnées

j indique l'ajout le plus récent à A

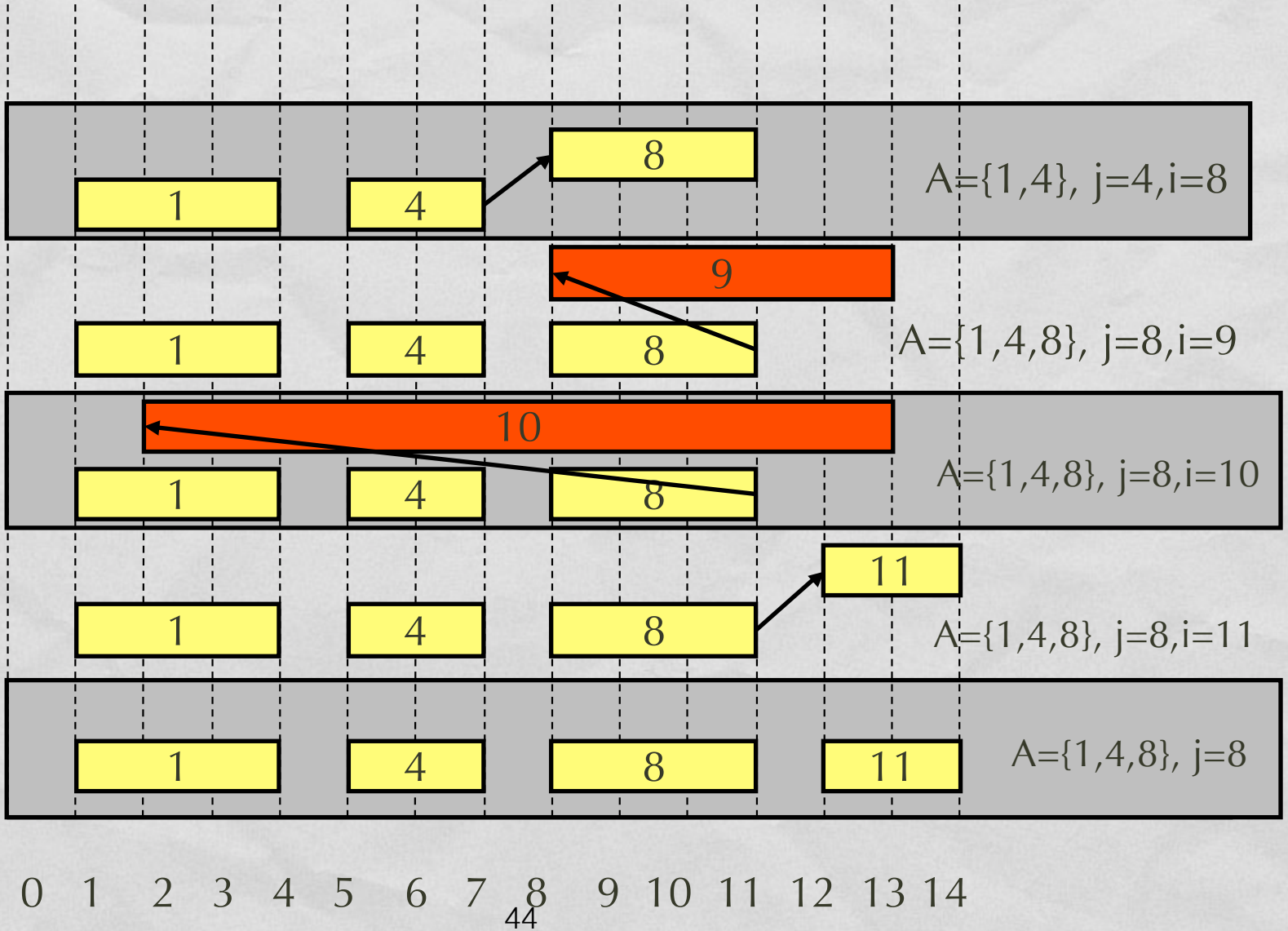
EXEMPLE

i	d	f
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	13



EXEMPLE

i	d	f
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	13



COMPLEXITÉ DE L'ALGORITHME GLOUTON

En supposant que les activités soient déjà triées selon leur horaire de fin

CHOIX-ACTIVITE-GLOUTON est très efficace

Elle peut ordonner un ensemble S de n activité en $\Theta(n)$

Rappel: Θ encadrant, il caractérise le fait que l'algorithme a des comportements similaires quelles que soient les données

VALIDITÉ DE L'ALGORITHME

CHOIX-ACTIVITE-GLOUTON

trouve toujours la solution optimale

Ce qui n'est pas le cas pour tous les algorithmes gloutons

THÉORÈME

L'algorithme de CHOIX-ACTIVITE-GLOUTON donne les solutions de la plus grande taille possible pour le problème du choix d'activité

PREUVE DU THÉORÈME

Supposons:

Soit S={1 , 2 , … , n} l'ensemble des activités ordonnancer.

Les activités sont supposées triées par l'horaire de fin

donc l'activité 1 possède

l'horaire de fin le moins tardif.

Montrons:

Qu'il existe une solution optimale

qui commence par un choix glouton,

c'est-à-dire par l'activité 1.

PREUVE DU THÉORÈME

Preuve: (piste)

Etablir que si une solution optimale A

au PCA commence par l'activité 1,

alors l'ensemble des activités A'=A-{1} est

une solution optimale du sous problème $S'=\{i \in S : d_i \geq f_1\}$

LES PIÈCES DE MONNAIE

Lélia Blin

Université d'Evry

LES PIÈCES DE MONNAIE

Considérons le problème consistant à rendre la monnaie

avec le moins de pièces possible.

Un commerçant aguerri applique l'algorithme glouton consistant

à rendre toujours la plus grosse pièce de valeur au plus ce qu'il reste à rendre.

Cela semble intuitivement la méthode la plus efficace

puisqu'à chaque étape

on fait le choix qui conduit à minimiser ce qu'il reste à rendre.

LES PIÈCES DE MONNAIE

En fait, si cette méthode marche pour les euros,

elle ne marche pas pour tous les systèmes !

LE SHILLING

Au début des années 70

Les monnaies anglaises comptaient encore la livre,

valant 20 shillings.

Il existait quatre subdivisions du shilling :

le penny (1/12 de shilling),

et trois pièces de valeurs respectives 1/4, 1/3, et 1/2 shillings,

soit respectivement 3, 4, et 6 pence (pluriel de penny).

LE SHILLING

Dans ce système rendre 8 pence conduit l'algorithme glouton à rendre

une pièce d'un demi shilling,

plus deux pièces d'un penny,

alors que la solution optimale consiste à rendre deux pièces d'un tiers de shilling,

puisque deux tiers de shilling font bien 8 pence!

LE SHILLING

Nous venons de voir un exemple où une approche gloutonne ne fonctionne pas

Les algorithmes gloutons se révèlent néanmoins très performants dans de nombreux contextes

Même outre Manche,

en tout cas depuis la réforme de décimalisation de 1971.

ELÉMENTS DE LA STRATÉGIE GLOUTONNE

Lélia Blin

Université d'Evry

ALGORITHME GLOUTON?

On a vu qu'un algorithme glouton peut conduire à une solution optimale

Mais que ce n'est pas toujours le cas

Comment peut-on savoir si un algorithme glouton saura résoudre un problème d'optimisation?

Il existe 2 caractéristiques qu'un problème peut avoir, et qui se prêtent à la stratégie gloutonne.

2 CARACTÉRISTIQUES

1er caractéristique:

Propriété du choix glouton:

on peut arriver à une solution optimale

en effectuant un choix localement optimal.

2eme caractéristique:

Sous structure optimale

un problème fait apparaître une sous structure optimale

si une solution optimale du problème contient

la solution optimale du sous problème. ex PCA

LE PROBLEME DU SAC À DOS

Lélia Blin

Université d'Evry

LE PROBLÈME DU SAC À DOS

Un voleur dévalisant un magasin trouve n objets

le ième objet vaut v_i euros et pèse w_i kilo

v_i et w_i sont des entiers

http://www.renders-graphiques.fr/

Il veut que son butin ait la plus grande valeur possible

mais il ne peut pas transporter plus de

W kilos dans son sac à dos (W entier)

Quels objets devra-t-il prendre pour maximiser son butin?

LE PB DU SAC À DOS: LES 2 VARIANTES

La variante « tout ou rien »:TOR

chaque objet doit être pris ou abandonné

le voleur ne peut pas prendre une partie de l'objet

ni prendre un objet plus d'une fois

La variante « fractionnaire »:Frac

le voleur peut prendre des fractions d'objets

STRATÉGIE GLOUTONNE

On doit calculer la valeur par kilo de chaque objet

prendre le plus possible de l'article ayant la plus grande valeur par kilo, et ainsi de suite

jusqu'à ce que le sac à dos soit plein

En triant les articles en fonction de leur valeur par kilo, on a un algorithme en O(n log n)

Cette stratégie peut résoudre Frac mais pas TOR,

bien que les problèmes soient similaires

SOUS-STRUCTURE OPTIMALE

Les 2 problèmes du sac à dos

satisfont la propriété de sous structure optimale

SOUS-STRUCTURE OPTIMALE

Pour TOR:

on considère le chargement de valeur max pesant au plus W kilos

si l'on retire l'objet j du sac

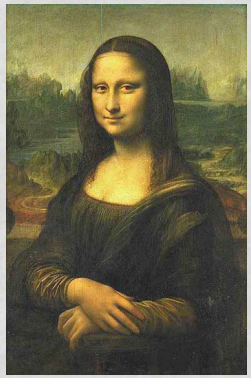
le chargement restant

doit être le meilleure que puisse prendre le voleur

pour un poids max $W-w_j$

à partir des n-1 objets initiaux,

j étant exclus



SOUS-STRUCTURE OPTIMALE

Pour Frac:

si l'on retire un poids w d'un objet j dans le chargement optimal,

le reste du chargement doit être

le meilleur que le voleur puisse prendre

pour un poids max de W-w

à partir des n-1 objets initiaux,

et des w_j-w kilos de l'objet j.

EXEMPLE DU « TOUT OU RIEN »

Il existe 3 types d'articles

article 1 pèse 10 kilos et vaut 60 € soit 6€/kg

article 2 pèse 20 kilos et vaut 100 € soit 5€/kg

article 3 pèse 30 kilos et vaut 120 € soit 4€/kg

EXEMPLE DU « TOUT OU RIEN »

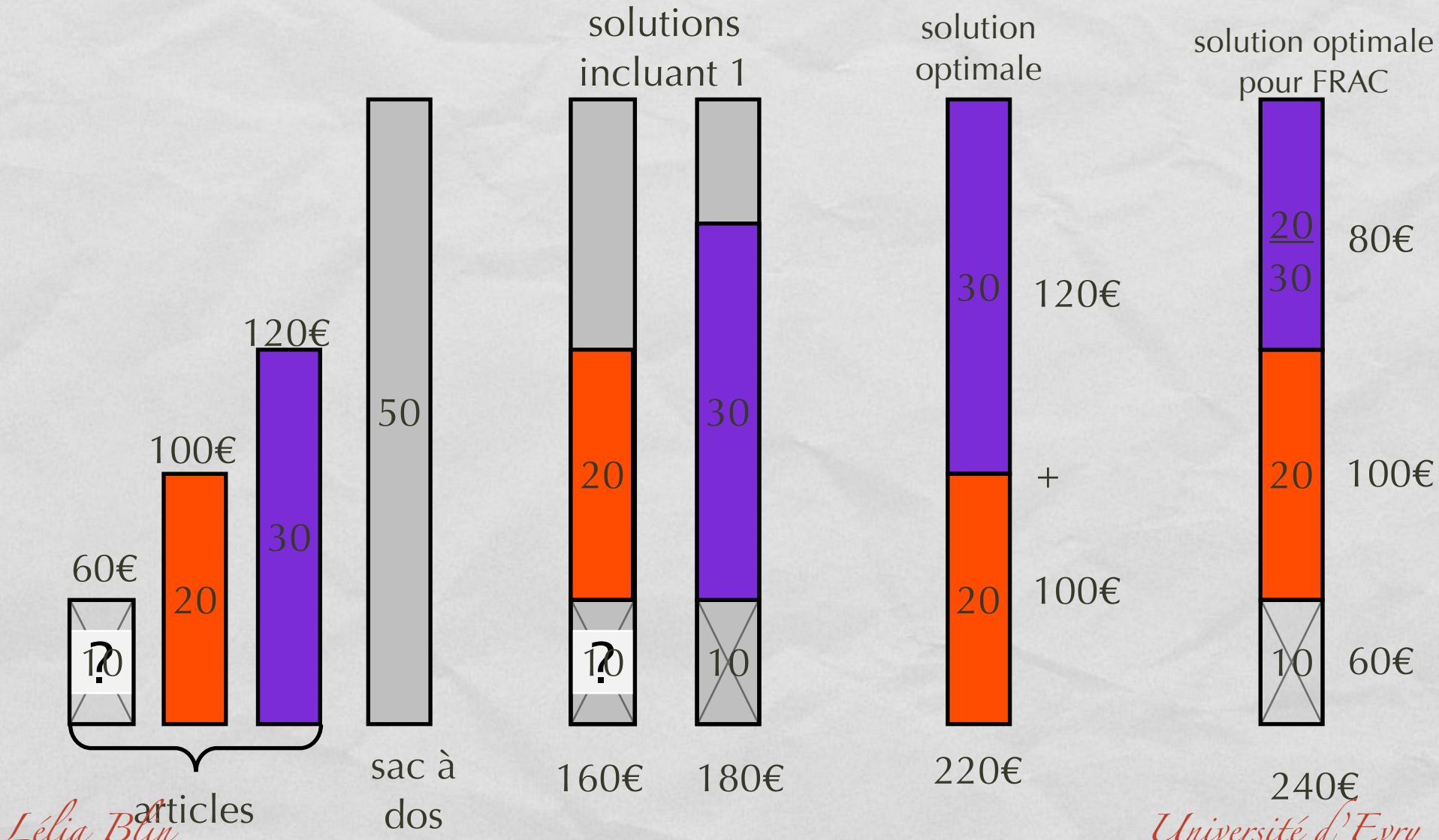
Le sac peut contenir 50 kilos au maximum

La stratégie gloutonne prendra

l'article ayant la plus grande valeur par kilo

soit l'article 1.

SOLUTION DE L'EXEMPLE



COMPRESSION DE DONNÉES

Lélia Blin

Université d'Evry

CODAGE DE HUFFMAN

Technique très efficace de compression de données

Principe:

Il utilise un tableau contenant les fréquences d'apparition de chaque caractère

Pour établir une manière optimale de représenter chaque caractère par une chaîne binaire

COMPRESSION DE DONNÉES: EXEMPLE

Un fichier de données de 100 000 caractères.

Le tableau représente la fréquence d'apparition de chaque caractère.

Ex: le caractère a apparaît 45 000 fois.

	a	b	c	d	e	f
Fréquence (en milliers)	45	13	12	16	9	5

CODAGE BINAIRE DES CARACTÈRES

Codage de longueur fixe

on a besoin de 3 bits pour représenter 6 caractères

on a donc besoin de 300 000 bits pour coder le fichier (de 100 000 caractères)

	a	b	c	d	e	f
Fréquence (en milliers)	45	13	12	16	9	5
Mot de code de longueur fixe	000	001	010	011	100	101

CODAGE BINAIRE DES CARACTÈRES

Codage de longueur variable

on attribut

aux caractères les plus fréquent les mots de codes courts

aux caractères les moins fréquent les mots de codes longs

Ici on a besoin de 224 000 bits pour le fichier

(45x1 + 13x3 + 12x3 + 16x3 + 9x4 + 5x4) x 1000 = 224 000 bits

	a	b	c	d	e	f
Fréquence (en milliers)	45	13	12	16	9	5
Mot de code de longueur fixe	000	001	010	011	100	101
Mot de code de longueur variable	0	101	100	111	1101	1100

CODAGE BINAIRE DES CARACTÈRES

Conclusion:

En faisant un codage de longueur variable

on fait une économie de 25%

par rapport à un codage de longueur fixe.

Cela représente un codage optimal

CODAGE ET DÉCODAGE PRÉFIXE

On considère que les codages où

aucun mot de code

n'est aussi préfixe d'un autre mot du code.

Ce type de codages sont dits préfixes.

CODAGE ET DÉCODAGE PRÉFIXE

On peut montrer

que la compression de données maximale accessible à l'aide d'un codage de caractères peut toujours être obtenue avec un codage préfixe ;

se restreindre aux codages préfixes ne fait donc pas perdre de généralité.

L'encodage est toujours simple pour n'importe quel code de caractères binaire ;

On se contente de concaténer les mots de code qui représentent les divers caractères du fichier.

CODAGE ET DÉCODAGE PRÉFIXE: EXEMPLE

Soit le codage préfixe de notre premier exemple.

On code abc par

0.101.100= 0101100

Décodage préfixe aucune ambiguïté possible, qu'une seule interprétation

0010111101

0. 0101111101 a

0. 0. 1011111101 aa

0. 0. 101. 1101 aab

	a	b	c	d	e	f
Fréquence (en milliers)	45	13	12	16	9	5
Mot de code de longueur variable	0	101	100	111	1101	1100

DÉCODAGE PRÉFIXE

Le processus de décodage exige

que le codage préfixe ait une représentation commode,

de manière qu'on puisse facilement repérer

le mot de code initial.

Une arborescence binaire

dont les feuilles sont les caractères donnés fournit ce genre de représentation.

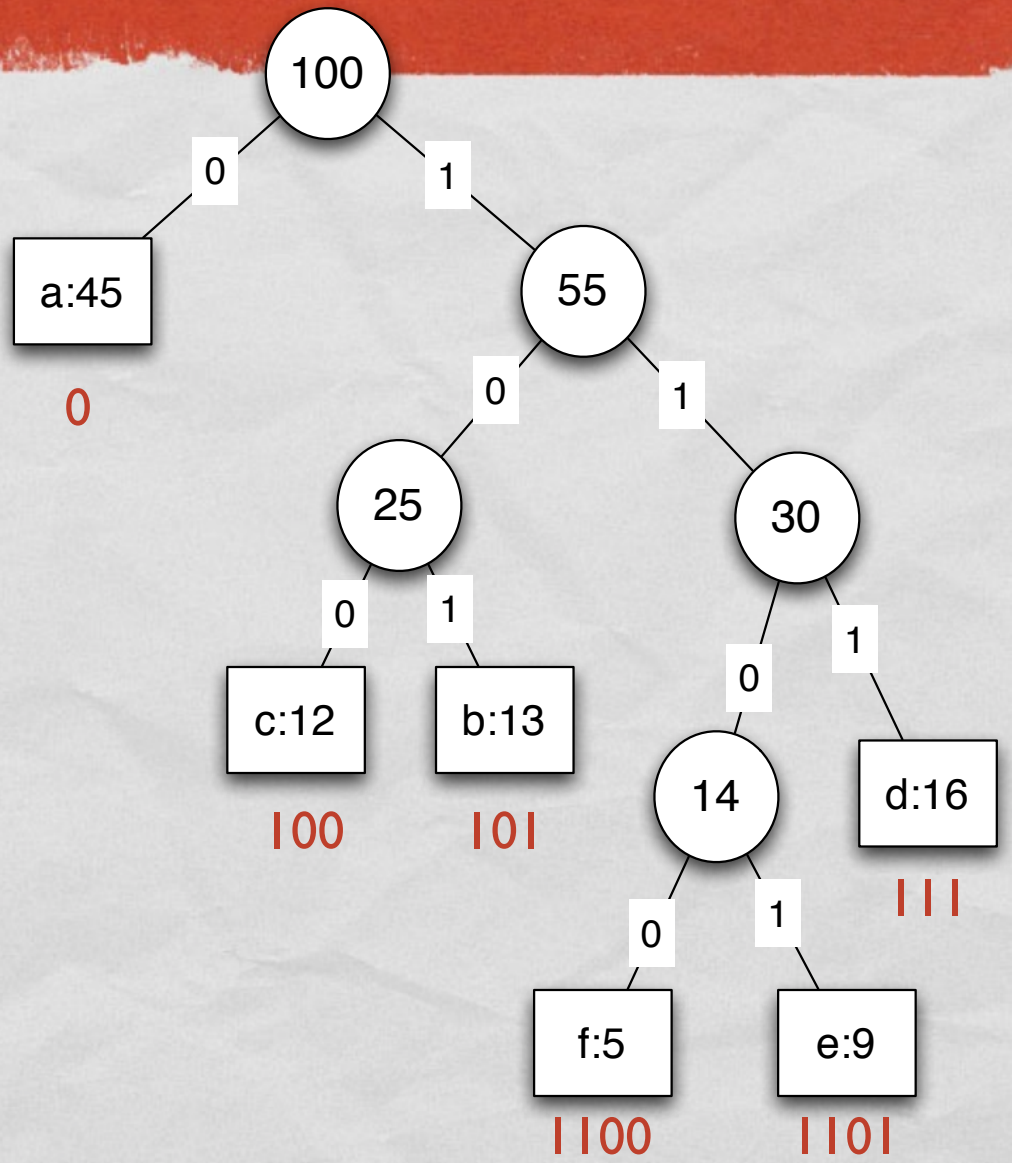
REPRÉSENTATION BINAIRE

On interprète le mot de code binaire associé à un caractère

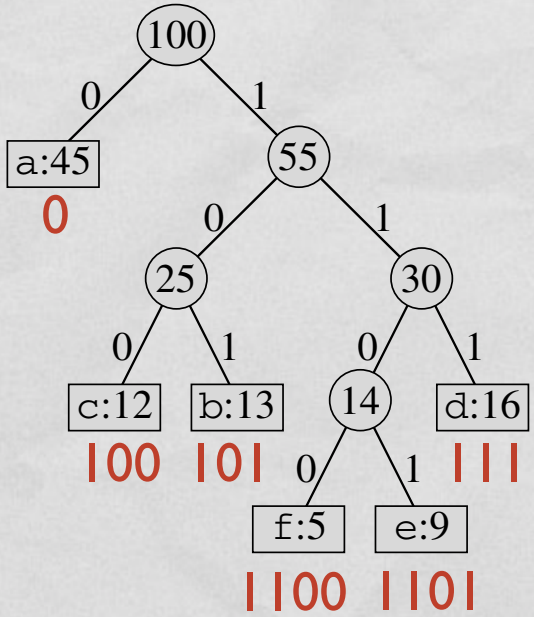
comme étant le chemin allant de la racine à ce caractère,

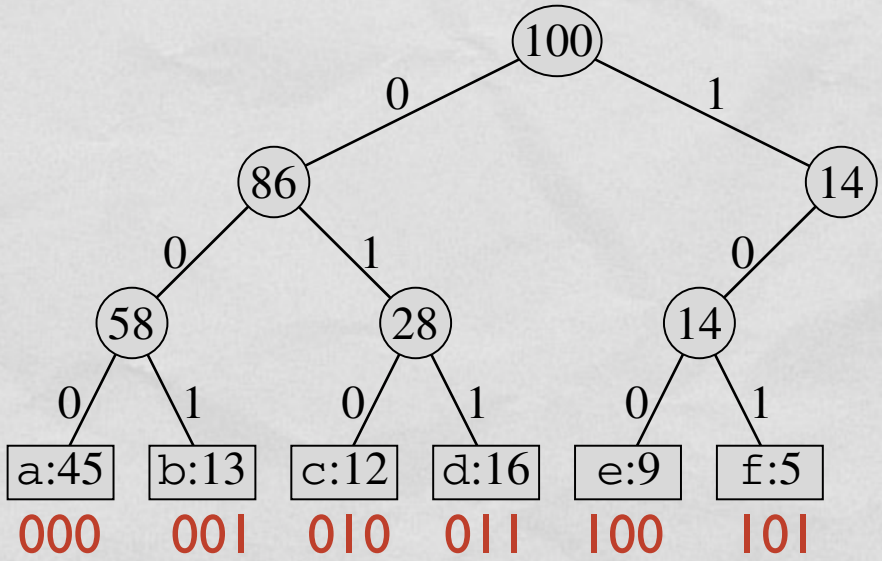
où 0 signifie « bifurquer vers l'enfant gauche »

et 1 signifie « bifurquer vers l'enfant droite ».



CODAGE OPTIMAL





CODAGE OPTIMAL

Un codage optimal pour un fichier est toujours représenté par:

Une arborescence binaire complète,

Dans laquelle

chaque nœud qui n'est pas une feuille a deux enfants .

Le codage de longueur fixe de notre exemple

n'est pas optimal puisque son arborescence

n'est pas une arborescence binaire complète

CODAGE OPTIMAL

On peut maintenant restreindre notre étude aux arbres binaires complets,

Soit C est l'alphabet d'où les caractères sont issus

On suppose que:

toutes les fréquences de caractère sont positives,

L'arborescence représentant un codage préfixe optimal possède

Exactement |C| feuilles, une pour chaque lettre de l'alphabet,

Et exactement |C| − 1 nœuds internes

CODAGE OPTIMAL PRÉFIXE

Étant donnée une arborescence T correspondant à un codage préfixe,

Pour chaque caractère c de l'alphabet C,

soit $f(c)$ la fréquence de c dans le fichier

et soit $d_T(c)$ la profondeur de la feuille c dans l'arbre.

Notez que $d_T(c)$ est aussi la longueur du mot de code pour le caractère c.

Le nombre de bits requis pour encoder un fichier vaut donc

$$B(T) = \sum_{c \in C} f(c) d_T(c) \,, \qquad (1)$$

ce qu'on définit comme étant le coût de l'arborescence T.

CODAGE DE HUFFMAN

Lélia Blin

Université d'Evry

CODAGE DE HUFFMAN: PSEUDO-CODE

```
HUFFMAN(C)
n ← |C|
F ← C
pour i← 1 à n-1 faire
   z ← ALLOUER-NŒUD()
   x ← gauche[z] ← EXTRAIRE-MIN(F)
   y ← droit[z] ← EXTRAIRE-MIN(F)
   f[z] ← f[x] + f[y]
   INSERER(F,z)
retourner EXTRAIRE-MIN(F)
```

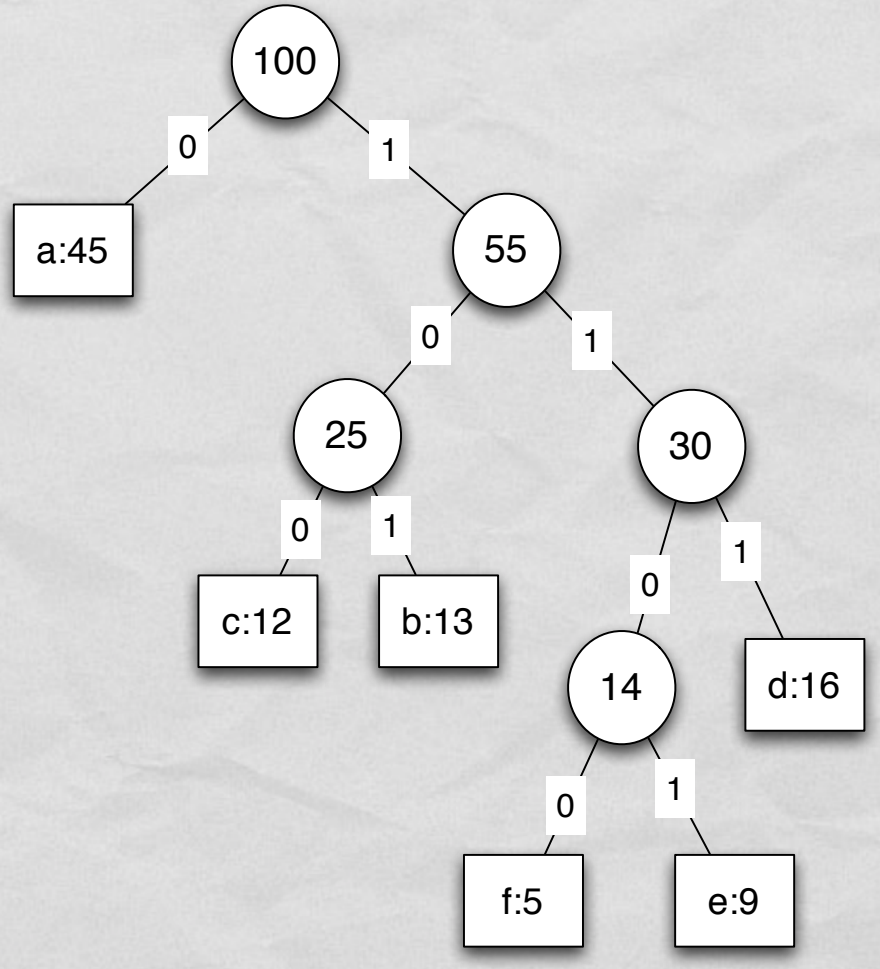
EXEMPLE

	a	b	c	d	e	f
Fréquence (en milliers)	45	13	12	16	9	5

i	gauche[z]	droit[z]	z	F
1	f:5	e:9	14	14,45,13,12,16
2	c:12	b:13	25	25,14,45,16
3	14	16:d	30	30,25,45
4	25	30	55	55,45
5	45:a	55	100	100

EXEMPLE

i	gauche[z]	droit[z]	z
1	f:5	e:9	14
2	c:12	b:13	25
3	14	16:d	30
4	25	30	55
5	45:a	55	100



AUTRE EXEMPLE

x	j	c	o	n	u	a	r	b
$f(x)$	7	12	30	14	20	45	2	10

Décoder le mot suivant:
100101001100010110010000

VALIDITÉ DE L'ALGORITHME DE HUFFMAN

Pour démontrer que l'algorithme glouton HUFFMAN est correct,

On montrera que le problème consistant à déterminer un codage préfixe optimal exhibe

les propriétés de choix glouton

et de sous-structure optimale.

Le lemme suivant montre que la propriété du choix glouton est respectée.

LEMME

Lemme:

Soit C un alphabet

Dans lequel chaque caractère $c \in C$ a une fréquence d'apparition $f[c]$.

Soient x et y deux caractères de C:

Ayant les fréquences d'apparition les plus basses.

Il existe alors un codage préfixe optimal pour C

Dans lequel les mots de code pour x et y

ont la même longueur

et ne diffèrent que par le dernier bit.

DÉMONSTRATION

Le principe de la démonstration est:

Prendre l'arborescence T représentant un codage préfixe optimal arbitraire

Le modifier pour en faire une arborescence

représentant un autre codage préfixe optimal

Tel que les caractères x et y apparaissent

comme des feuilles sœur

de profondeur maximale dans la nouvelle arborescence.

DÉMONSTRATION

Si on peut faire cela, alors

leurs mots de code auront la même longueur

et ne différeront que par le dernier bit.

DÉMONSTRATION

Soient a et b les deux caractères qui sont

des feuilles sœurs de profondeur maximale dans T.

Sans perdre le caractère général de la démonstration, on suppose que

$f[a] \leq f[b]$ et $f[x] \leq f[y]$.

Comme $f[x]$ et $f[y]$ sont les deux fréquences de feuille les plus basses,

dans l'ordre,

et que $f[a]$ et $f[b]$ sont deux fréquences arbitraires,

également dans l'ordre,

on a $f[x] \leq f[a]$ et $f[y] \leq f[b]$.

DÉMONSTRATION

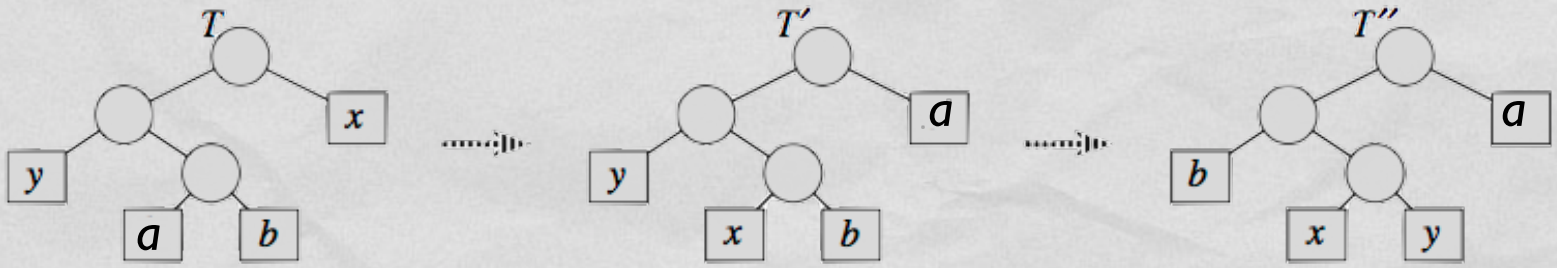
Comme illustré ci-dessous:

On permute les positions dans T de a et x

pour produire une arborescence T′

Puis on permute les positions dans T′

de b et y pour produire une arborescence T′′.



DÉMONSTRATION

D'après l'équation

$$B(T) = \sum_{c \in C} f(c) d_T(c) \ ,$$

la différence de coût entre T et T′ est

$$\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_{T'}(x) - f[a] d_{T'}(a) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_T(a) - f[a] d_T(x) \\
&= (f[a] - f[x])(d_T(a) - d_T(x)) \\
&\geqslant 0 \ ,
\end{aligned}$$

DÉMONSTRATION

On obtient ce résultat

car f [a] − f [x] et d_T [a] − d_T [x] sont tous deux positifs ou nuls.

Plus précisément, f [a] − f [x] est positif ou nul

parce que x est une feuille de fréquence minimale,

et d_T [a] − d_T [x] est positif ou nul

parce que a est une feuille de profondeur maximale dans T.

DÉMONSTRATION

De même, comme la permutation de y et b

n'augmente pas le coût,

$B(T') - B(T'')$ est positif ou nul.

Donc, $B(T'') \leq B(T)$,

et comme T est optimal, $B(T) \leq B(T'')$,

ce qui implique $B(T'') = B(T)$.

Donc, T'' est une arborescence optimale

dans laquelle x et y sont des feuilles sœur de profondeur maximale,

ce qui prouve le lemme.

CONCLUSION DU LEMME

Le lemme implique que

le déroulement de la construction d'une arborescence optimale par fusions successives

peut, sans perte de généralité,

commencer par le choix glouton consistant à fusionner les deux caractères ayant les fréquences les plus faibles.

Pourquoi est-ce le choix glouton ?

On peut voir le coût d'une simple fusion comme la somme des fréquences des deux éléments fusionnés.

Le lemme suivant montre que le problème de la construction d'un codage préfixe optimal vérifie la propriété de sous-structure optimale.

LEMME

Lemme:

Soit C un alphabet donné, avec une fréquence f [c] définie pour chaque caractère c $\in$ C.

Soient x et y deux caractères de C ayant la fréquence minimale.

Soit C′ l'alphabet C privé des caractères x,y

et complété par le (nouveau) caractère z,

de sorte que C′ = C − {x,y} ∪ {z};

définissons f pour C′ comme pour C, sauf que f [z] = f [x] + f [y].

Soit T ′ une arborescence représentant un code préfixe optimal pour l'alphabet C′.

Alors, l'arborescence T, obtenue à partir de T′

en remplaçant le nœud feuille associé à z

par un nœud interne ayant x et y comme enfants,

représente un code préfixe optimal pour l'alphabet C.

DÉMONSTRATION

On commence par montrer que le coût B(T) de l'arborescence T

peut être exprimé en fonction du coût B(T′) de l'arborescence T′

en considérant les coûts qui le composent dans l'équation(1).

Pour chaque $c \in C - \{x, y\}$,

on a $d_T(c) = d_{T'}(c)$,

et donc $f[c]d_T(c) = f[c]d_{T'}(c)$.

Puisque $d_T(x) = d_T(y) = d_{T'}(z)+1$, on a

$$\begin{aligned} f[x]d_T(x) + f[y]d_T(y) &= (f[x] + f[y])(d_{T'}(z) + 1) \\ &= f[z]d_{T'}(z) + (f[x] + f[y]) , \end{aligned}$$

d'où l'on conclut que

$$B(T) = B(T') + f[x] + f[y] .$$

ou, ce qui revient au même,

$$B(T') = B(T) - f[x] - f[y] .$$

DÉMONSTRATION

Nous allons maintenant prouver le lemme en raisonnant par l'absurde.

Si T ne représente pas un codage préfixe optimal pour C,

il existe une arborescence T′′ tel que B(T′′) < B(T).

Sans nuire à la généralité (d'après le lemme), T′′ a x et y comme frères.

Soit T′′′ l'arborescence T′′ dans laquelle

le parent commun à x et à y a été remplacé par

une feuille z de fréquence f [z] = f [x] + f [y].

Alors

$$\begin{aligned} B(T''') &= B(T'') - f[x] - f[y] \\ &< B(T) - f[x] - f[y] \\ &= B(T') , \end{aligned}$$

Ce qui contredit l'hypothèse que T′ représente un codage préfixe optimal pour C′.
Donc, T représente forcément un codage préfixe optimal pour l'alphabet C.

THÉORÈME

L'algorithme de HUFFMAN produit un codage préfixe optimal.

Démonstration :

Immédiate d'après les deux lemmes précédents.

BIBLIOGRAPHIE

Certaine partie de ce cours sont tiré du livre Introduction à l'Algorithmique