6
ALGORITHMES GLOUTONS

PLAN

- **Problèmes d'optimisation**
- **Algorithmes gloutons**
- **Arbre couvrant minimal**
- **Choix d'activités**
- **Codage de Huffman**
- **Matroïdes**

6.1 PROBLEMES D ’OPTIMISATION

un problème → plusieurs solutions

une solution → une valeur

problème d'optimisation:

recherche d'une solution de valeur optimale (min ou max)

algorithme de résolution:

- **reconnaissance d'une solution**
- **évaluation d'une solution**
- **sélection d'une des meilleures solutions**

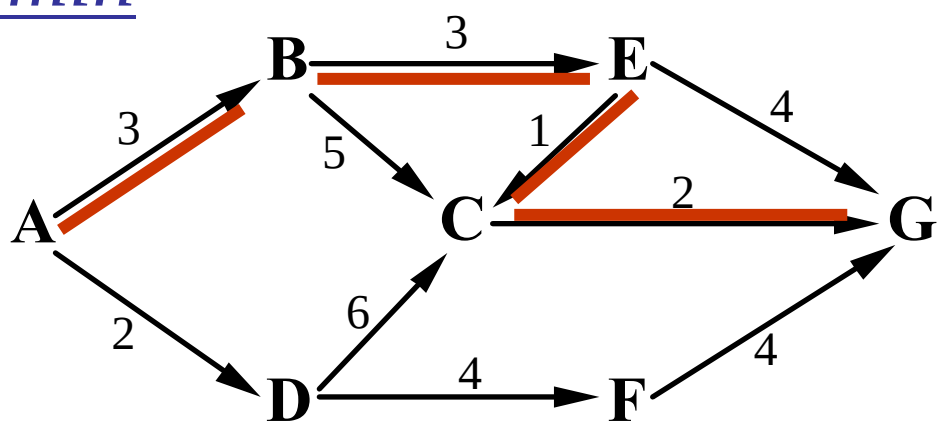
problèmes d'optimisation

→ **faciles**

→ **difficiles**

EXEMPLES

E1) chemin min



problème facile (v=9;ABECG)

E2) sac-à-dos (en nombres entiers)

problème "assez" difficile

aliments	A	B	C
poids unitaire hg	6	4	3
valeur nutritive unitaire	14	10	6
poids max des aliments: 17			

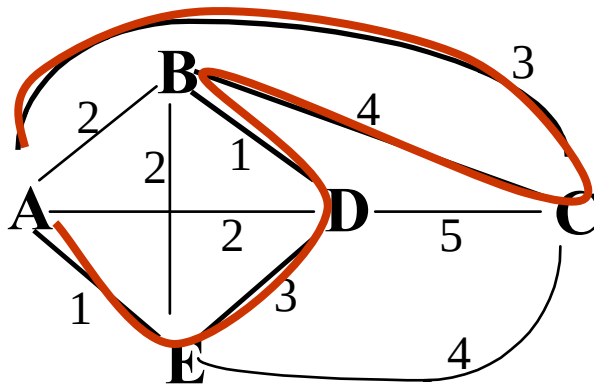
2 solutions optimales:

1.A + 2.B + 1.C $\rightarrow$ **v = 40 (p=17)**

4.B $\rightarrow$ **v = 40 (p=16)**

E3) voyageur de commerce

problème difficile



ici, 2 solutions optimales

ACBDEA et ACEBDA **v = 12**

nombre de solutions: 24 (n-1!)

---

problème de petite taille

→ énumération possible

problème de grande taille

→ énumération impossible

INTERDIT D'ÉNUMÉRER DANS LE COMBINATOIRE

optimisation discrète ≠ optimisation continue

problèmes faciles $\rightarrow$

algorithme de complexité polynomiale

(cf cours 2)

problèmes "difficiles" $\rightarrow$

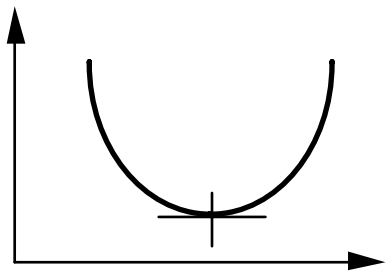
- **tous les algorithmes connus sont de complexité exponentielle**
- **problèmes dits NP-complets ou NP-difficiles**

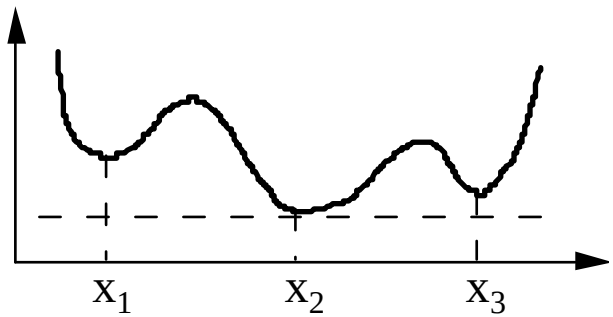
cours VARI:problèmes "faciles"

6.2 ALGORITHMES GLOUTONS

choix glouton = choix localement optimal

optimum local ≠ optimum global





fonction concave (ou convexe)
optimum local = optimum global

continu ≠ entier

x_1 et x_3 : optima locaux

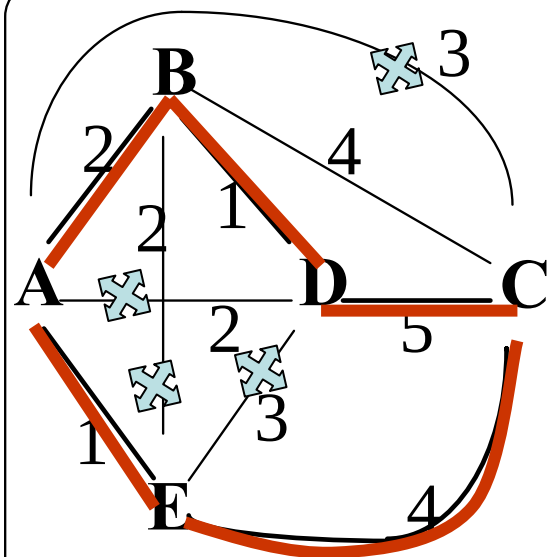
x_2 optimum global (et local)

algorithme glouton: pas toujours optimal

EXEMPLE

un algo glouton pour le PVC:

- **trier les arêtes (coûts croissants)**
- **sélectionner dans l'ordre les arêtes non "parasites"**



([A,E],[B,D],[A,B],
~~[A,D]~~,~~[B,E]~~,~~[A,C]~~,
~~[E,D]~~,[C,E],[C,D])

[A,B,D,C,E,A]
valeur 13

(opt=12 [ACEBDA])

algorithme glouton: à chaque étape
choix le plus intéressant à cet instant

→ facile à concevoir

→ difficile de vérifier l'optimalité

→ efficace (faible complexité)

6. 3 ARBRE COUVRANT MINIMAL

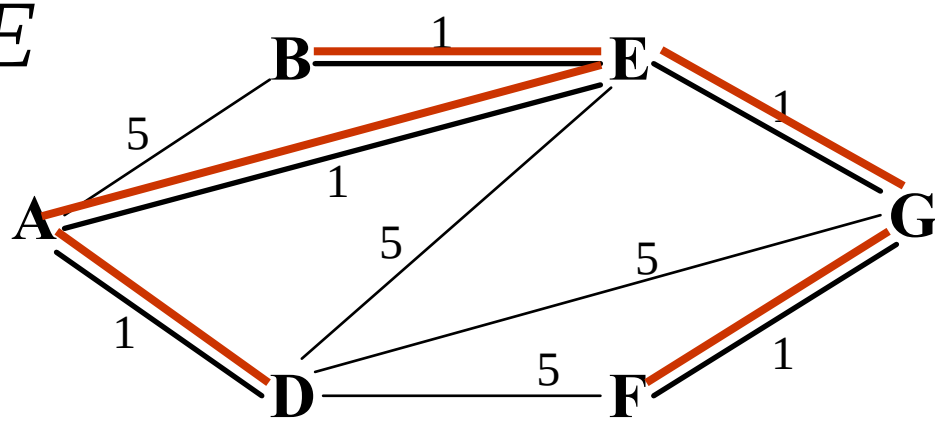
6.3.1 LE PROBLÈME

relier des objets avec une
longueur totale minimale des liens

graphe valué associé:

- **sommets = objets**
- **arête = lien possible**
- **poids d'une arête = longueur du lien**

EXEMPLE



solution optimale:

sans cycle et connexe → ARBRE

par tous les sommets → COUVRANT

de longueur totale min → MINIMAL

6.3.2 ALGORITHME DE KRUSKAL

rappel: arbre $\rightarrow$ m = n-1 arêtes

entrée: **G = (X,U,P)**

n sommets, m arêtes

sortie: **A arbre couvrant min de G**

void **kruskal** (in **G**: **graphe;** out **A: arbre):**

entier k=0; {arêtes} V=∅;

début

trier les arêtes de G par ordre de poids croissant ;

tant que **k < n-1** faire

parcourir la liste triée ;

sélectionner la première arête qui ne forme pas de cycle avec les précédentes, w ;

k = k+1 ;

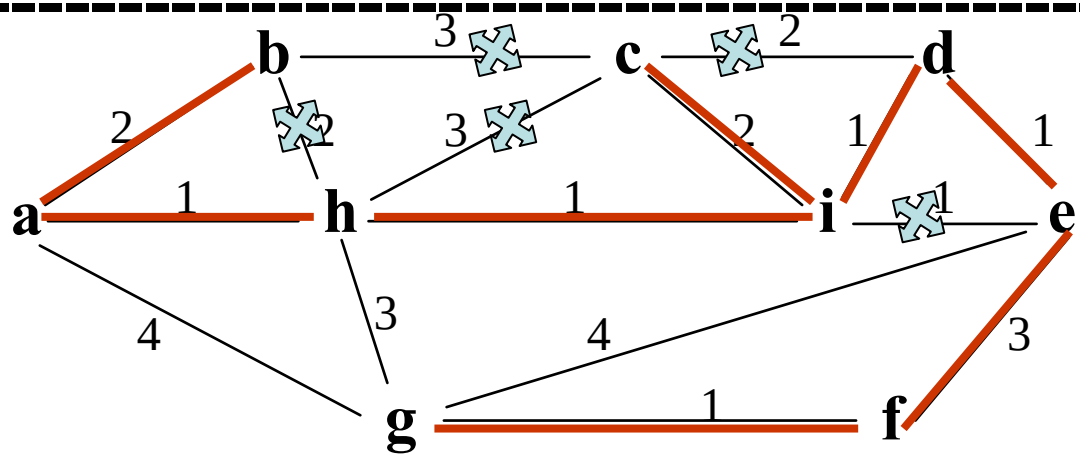
V = V ∪ {w} ;

fait ;

A = graphe (X,V) ;

fin

EXEMPLE



liste triée:

-

ah	de	di	ei	hi	fg	ab	bh	ci	cd	bc	ch	ef	gh	ag	ge
1	1	1	1	1	1	2	2	2	2	3	3	3	3	4	4
x	x	x	-	x	x	x	-	x	-	-	-	x	STOP		

n = 9 → stop après 8 sélections

P(A) = (1+1+1+1+1+2+2+3) = 12

- **implémentation peu facile**

  (sera faite en exercices dirigés)

- **complexité O(m logm)** (TRI)

CArbre **kruskal ( )**

entier k =0; {arêtes} V=∅;

début

trier les arêtes de G par ordre de poids croissant ;

tant que **k < n-1** faire

parcourir la liste triée ;

sélectionner la première arête qui ne forme pas de cycle avec les précédentes, w ;

k = k+1 ;

V = V ∪ {w} ;

fait ;

Retourner A = graphe (X,V) ;

fin

6.3.3 PREUVE DE L 'OPTIMALITE

G=(X,U) le graphe

A=(X,V) l'arbre couvrant obtenu

p(u) : poids de l'arête u

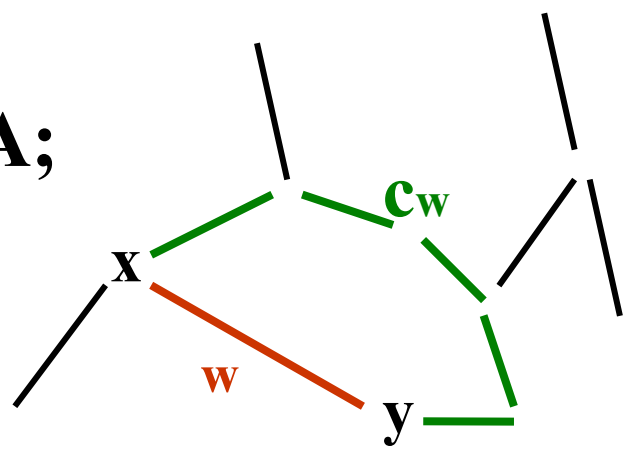
propriété 1

soit w ∈ U -V, w=[x,y] et

c_w : chaîne de x à y dans A;

alors,

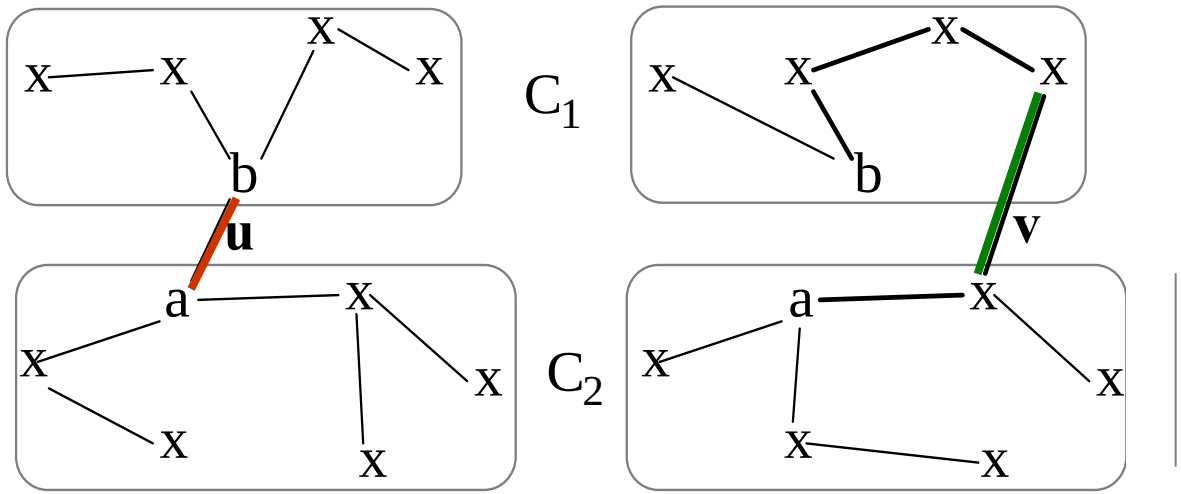
$p(w) \geq \max_{u \in cw} p(u)$



A': arbre optimal de poids p(A')

montrons que p(A) = p(A')

soit u ∈ A'-A



A'=(X,V') **A=(X,V)**

u ∈ V'-V **relie C_1 et C_2 dans A'**

v ∈ c_u **relie C_1 et C_2 dans A**

propriété 1 $\Rightarrow$ $p(u) \geq p(v)$ **et**

A' minimal $\Rightarrow$ $p(v) \geq p(u)$

$\Rightarrow$ $p(u) = p(v)$

soit A"=A'+{v}-{u}: p(A")=p(A')

$\rightarrow$ **A"optimal et** $v \in A"$

soit u'$\in$ A"-A,... (idem... $\rightarrow$ A'")

...

fin: A'"" = A et $p(A'"") = p(A') = p(A)$

6. 4 CHOIX D 'ACTIVITES

6.4.1 LE PROBLÈME

$A = \{a_1, a_2, .., a_n\}$ activités

$a_i \in [d_i, f_i[$ d_i début, f_i fin de a_i

a_i et a_j compatibles si $d_i \geq f_j$ ou $d_j \geq f_i$

problème:

choisir le plus grand nombre d'activités compatibles

6.4.2 ALGORITHME CHOIX_ACTIVITÉ

choix glouton : maximiser le temps restant après l'activité choisie

Algorithme **choix_activité** **(**in **A**: **liste initiale d'activités**;
out **A*: liste d'activités choisies):**

indice j ;
début
trier et numéroter les a_i dans l'ordre croissant des f_i ; $f_1 \leq f_2 \leq \ldots \leq f_n$
$A^* = \{a_1\}$;
j = 1; *j représente la dernière activité sélectionnée*
pour **i = 1** à **n** faire
    si **$d_i \geq f_j$** alors
        $A^* = A^* \cup \{a_i\}$;
        j = i ;
    finsi;
fait ;
fin

complexité: $O(n\log n)+O(n)$

tri *boucle*

$\Rightarrow O(n\log n)$

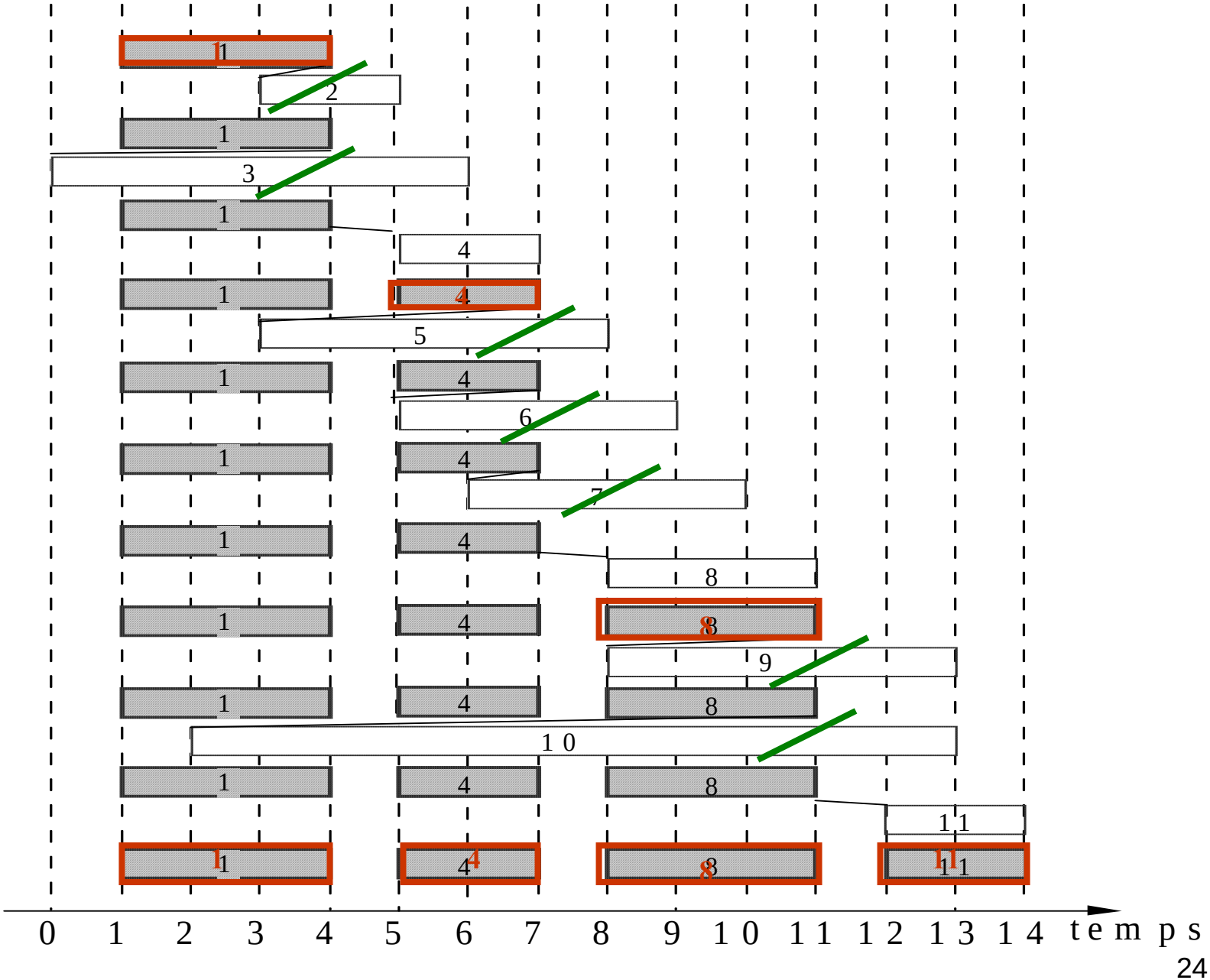
EXEMPLE

d_i date de début de l'activité a_i

f_i date de fin de l'activité a_i

activités triées selon les f_i

i	1	2	3	4	5	6	7	8	9	10	11
d_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14


1
2
1
3
1
4
1
4
5
1
4
6
1
4
7
1
4
8
1
4
8
9
1
4
8
1 0
1
4
8
1 1
1
4
8
1 1
0 1 2 3 4 5 6 7 8 9 1 0 1 1 1 2 1 3 1 4
t e m p s

6.4.3 OPTIMALITE

Soit B solution optimale $B \neq A^*$ card $B \geq$ card A^*

B ordonné selon les f_i:

$$a_{k1}, a_{k2}, \ldots, a_{kn} \text{ avec } f_{k1} \leq f_{k2} \leq \ldots \leq f_{kn}$$

si $a_{k1} \neq a_1$ **, soit** $B' := B - \{a_{k1}\} + \{a_1\}$

- $f_1 \leq f_{k1} \leq f_{k2} \leq \ldots \leq f_{kn}$ $\rightarrow$ B' admissible
- cardB' = cardB $\Rightarrow$ B' optimal

sinon B':= B

il existe une solution optimale commençant par un choix glouton

problème restant = problème initial avec

$$A' = \{a_i \in A \text{ t.q. } d_i \geq f_1\}$$

$\rightarrow$ **choix glouton suivant de B': a_2**

...

fin: $B''''' = A^*$

et card $B''''' =$ card B = card A^*

6. 5 CODAGE DE HUFFMAN

compression de données

gain de place de 20 à 90%

C = {caractères}

$c_i \in C \rightarrow$ chaîne binaire unique

EXEMPLE

C = alphabet

d-01 f-101 m-0110 ...

dfd = 0110101

MAIS mf = 0110101 (= dfd !)

---

codage préfixe : aucun mot de code n'est préfixe d'un autre mot

décodage facile

EXEMPLE

caractère	a	b	c	d	e	f
code	000	001	010	011	100	101

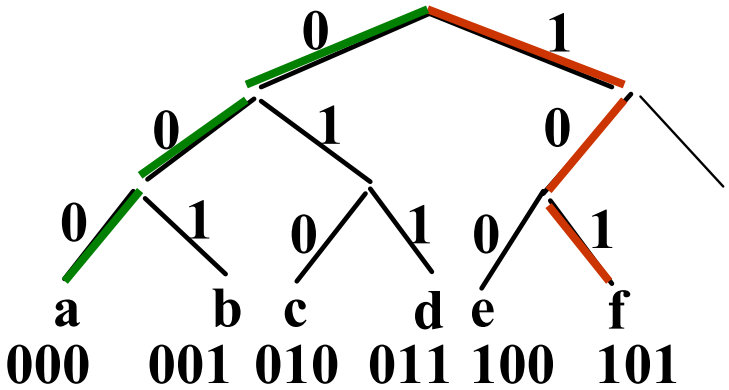
---

face → 101000010100 → face

concevoir le codage et décoder efficacement

arbre associé (binaire et complet)

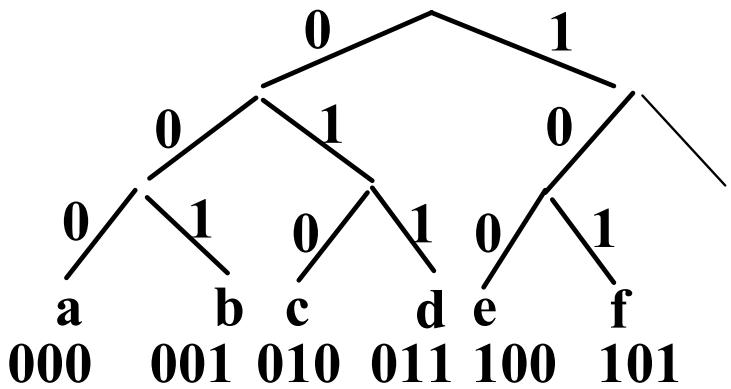
A



décodage: 101000010100 ← **f a c e**

arbre associé (binaire et complet)

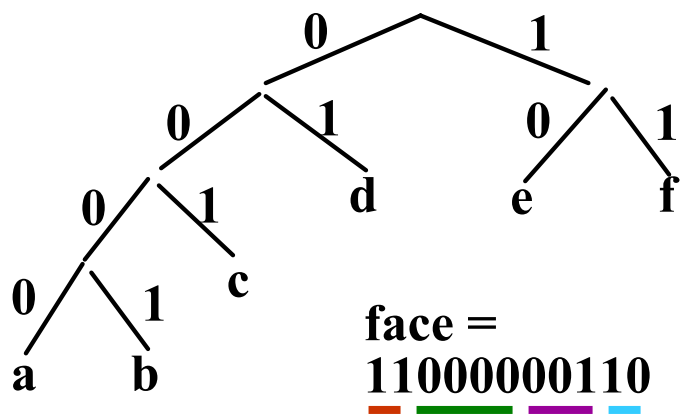
A



décodage: 101000010100 ← face

autre possibilité

A’



choix d'un arbre

tableau des fréquences d'apparition des caractères dans le fichier $(c \longrightarrow f(c))$

$h_A(c)$: **hauteur de la branche c**

$=$ **nombre de bits pour c**

nombre de bits requis pour encoder un fichier selon l'arbre A

$$B(A) = \sum_{c \in C} f(c)\, h_A(c)$$

EXEMPLE

	a	b	c	d	e	f
fréq.	45	13	12	16	9	5
code A'	000	001	010	011	100	101
code A''	0000	0001	001	01	10	11
code H	0	101	100	111	1101	1100

B(A') = 300 (45+13+12+16+9+5)*3

B(A'')= 328 (45*4+13*4+12*3+16*2+9*2+5*2)

B(H) = 224 (45*1+13*3+12*3+16*3+9*4+5*4)

code de Huffman

caractères fréquents $\rightarrow$ mots courts

nombre de feuilles = card(C)

feuille $\rightarrow$ c$\in$C $\rightarrow$ "val"= f(c)

"val" d'un nœud interne	**=**	**somme des "val" des 2 fils**

Algorithme **HUFFMAN** (in **C: liste de caractères**; in **f: C → N**;
out **A: arbre de Huffman):**

avec void **construire(z,x,y)** *qui crée un nœud z ayant pour fils gauche x et pour fils droit y*

liste_ordonnée L ; entier n;

début

n = card(C) ;

L = C;

tri de L selon les f(c) croissants;

pour **i = 1** à **n-1** faire

x = L.min; L.supprimer_min;

y := L.min; L.supprimer_min;

construire (z,x,y);

f(z) = f(x) + f(y);

L.insérer (z);

fait ;

complexité: O(nlogn)

fin

6. 5 MATROIDES

complément facultatif du cours

6.5.1 DÉFINITION

$E = \{e_1, e_2, .., e_n\}$ $\quad I \subset \mathcal{P}(E)$

matroïde: couple M = (E,I) t.q.

- $F \in I$ et $F' \subset F \Rightarrow F' \in I$
- $\forall S \subset E$,

  [F ∈ I et F' ∈ I et F , F' sous-ensembles maximaux de S]

  ⇒ card F = card F ’

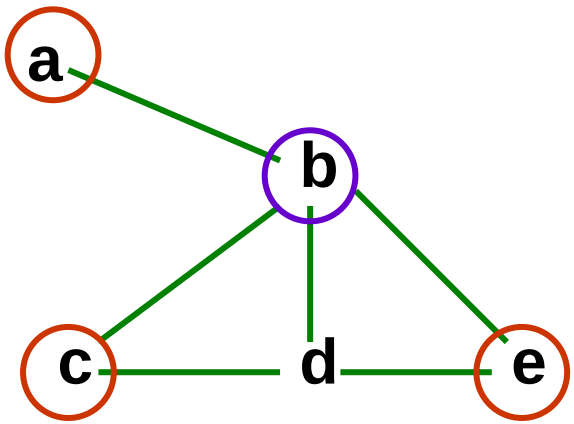
I famille de sous-ensembles « indépendants »

$$\varnothing \in I \text{ et } \forall i, \{e_i\} \in I$$

Contre-exemple:

E= ensemble des sommets d'un graphe

I = ensemble des ensembles stables



S=Ensemble Stable=sous-ensemble de sommets du graphe tel que deux sommets de S ne sont pas voisins. Tout ensemble inclus dans S est donc stable. S est maximal si l'ajout d'un sommet quelconque le rend non stable.

{a,c,e}, {a,d} et {b}

Sont des stables maximaux qui n'ont pas le même cardinal.

EXEMPLE

graphe $G=(X_G,E_G)$

M_G matroïde graphique

$M_G = (E_G,I_G)$

$I_G = \{F \subset E_G$ t.q. $G'=(X_G,F)$ sans cycle$\}$

sous-ensembles maximaux = arbres couvrants

même cardinal: n-1 arêtes

6.5.2 MATROIDES PONDERES

$M = (E, I, w)$

$e \in E$ $\quad w_e$ = poids de e

$F \subset E$ $\quad w(F) = \sum_{e \in F} w_e$

= poids de F

problème:

trouver $F \subset I$ t.q. w(F) minimal (ou maximal)

algorithme **glouton_min** (in **M = (E,I,w): matroïde**;
out **F: élément de I):**

début

F = ∅;

trier et numéroter les éléments de E par ordre de poids croissant;

pour **i = 1** à **n** faire

si **F∪{e_i} ⊂ I** alors

F = F∪{e_i} ;

finsi

fait ;

fin

théorème

M matroïde pondéré :

l'algorithme glouton donne toujours l'optimum

complexité

si test en O(f(n))

O(nlogn) + O(nf(n))

tri *boucle pour*

(dépend de f(n))