

Module EA4 – Éléments d'Algorithmique II

Outils pour l'analyse des algorithmes

Dominique Poulalhon

`dominique.poulalhon@irif.fr`

Université Paris Diderot

L2 Informatique & Math-Info

Année universitaire 2016-2017

CETTE SEMAINE...

Les TD seront remplacés par des TP pour les groupes INFO :

- INFO 1 : 442C
- INFO 2 : 432C
- INFO 3 : 436C
- INFO 4 : 449C

Pour les Math-Info, rendez-vous dans la salle de TD habituelle

D'AUTRES ARBRES « TRIÉS » : LES TAS

tas-min et tas-max

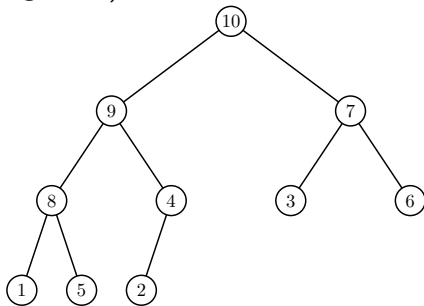
arbre binaire « presque parfait » tel qu'en chaque nœud, l'étiquette est inférieure (*resp.* supérieure) à celles de ses fils

D'AUTRES ARBRES « TRIÉS » : LES TAS

tas-min et tas-max

arbre binaire « presque parfait » tel qu'en chaque nœud, l'étiquette est inférieure (*resp.* supérieure) à celles de ses fils

arbre binaire presque parfait : dont tous les niveaux sont totalement remplis sauf éventuellement le dernier (qui est rempli depuis la gauche)



QUEL EST L'INTÉRÊT DES TAS(-MAX) ?

accéder en temps **constant** à l'élément (de priorité) maximal(e)

QUEL EST L'INTÉRÊT DES TAS(-MAX) ?

accéder en temps **constant** à l'élément (de priorité) maximal(e)
– à la racine

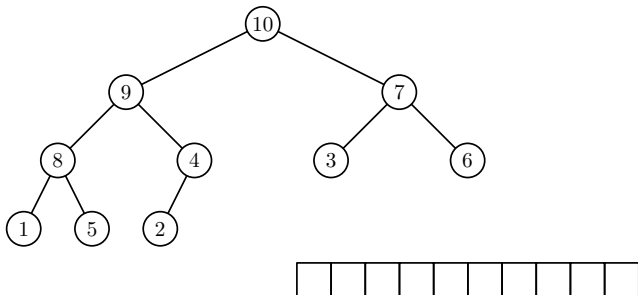
QUEL EST L'INTÉRÊT DES TAS(-MAX) ?

accéder en temps **constant** à l'élément (de priorité) maximal(e)
– à la racine

hauteur optimale : $\log n$ (ou plus exactement $\lfloor \log n \rfloor$)

QUEL EST L'INTÉRÊT DES TAS-MAX ?

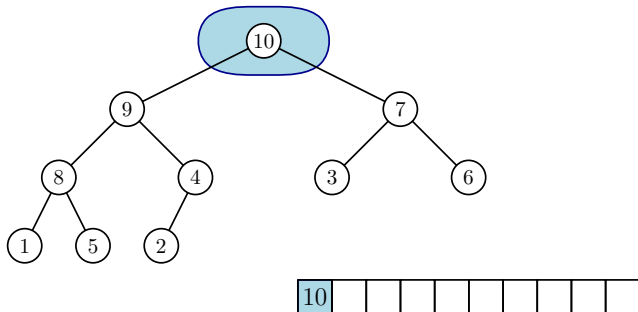
et surtout, **très facile à représenter** par un tableau :



QUEL EST L'INTÉRÊT DES TAS-MAX ?

et surtout, **très facile à représenter** par un tableau :

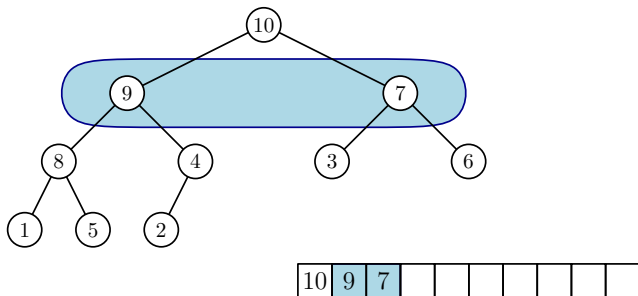
- stocker les nœuds dans l'ordre du parcours en largeur



QUEL EST L'INTÉRÊT DES TAS-MAX ?

et surtout, **très facile à représenter** par un tableau :

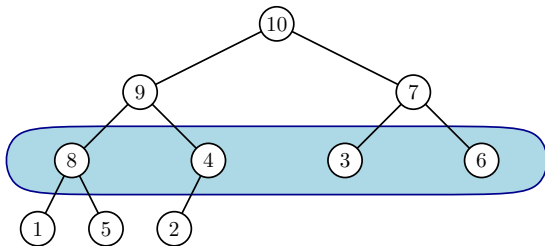
- stocker les nœuds dans l'ordre du parcours en largeur



QUEL EST L'INTÉRÊT DES TAS-MAX ?

et surtout, **très facile à représenter** par un tableau :

- stocker les nœuds dans l'ordre du parcours en largeur

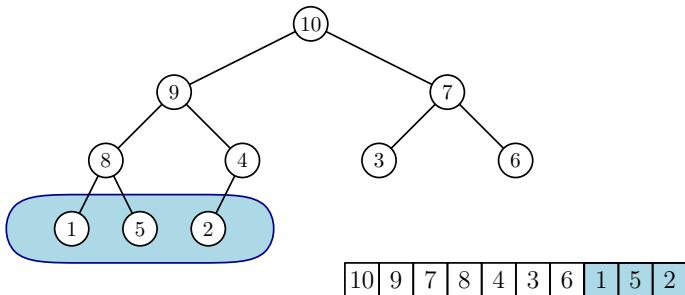


10	9	7	8	4	3	6			
----	---	---	---	---	---	---	--	--	--

QUEL EST L'INTÉRÊT DES TAS-MAX ?

et surtout, **très facile à représenter** par un tableau :

- stocker les nœuds dans l'ordre du parcours en largeur

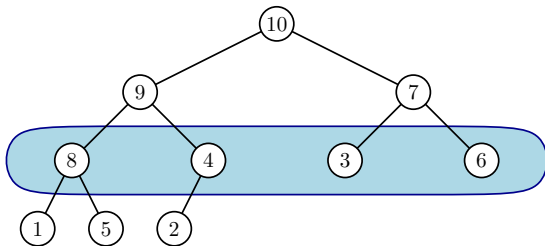



QUEL EST L'INTÉRÊT DES TAS-MAX ?

et surtout, **très facile à représenter** par un tableau :

- stocker les nœuds dans l'ordre du parcours en largeur
- le niveau h est stocké entre les positions 2^h et $2^{h+1} - 1$

(convention pratique : les positions commencent à 1)



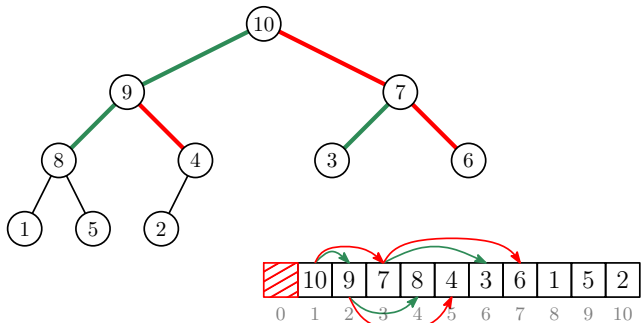
	10	9	7	8	4	3	6	1	5	2
0	1	2	3	4	5	6	7	8	9	10

QUEL EST L'INTÉRÊT DES TAS-MAX ?

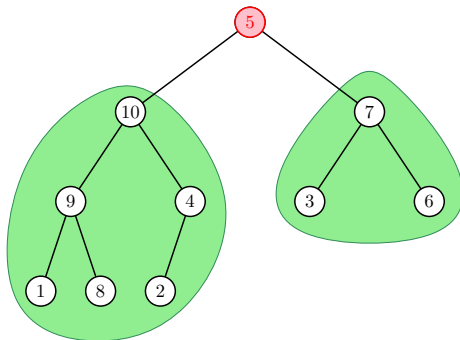
et surtout, **très facile à représenter** par un tableau :

- stocker les nœuds dans l'ordre du parcours en largeur
- le niveau h est stocké entre les positions 2^h et $2^{h+1} - 1$
- $\text{pere}(i) = \lfloor i/2 \rfloor$, $\text{gauche}(i) = 2i$, $\text{droit}(i) = 2i + 1$

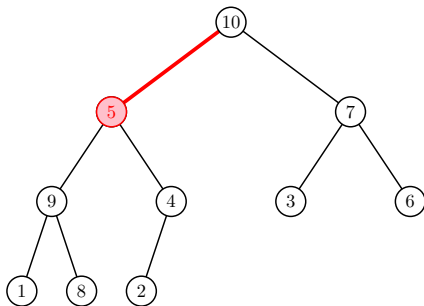
(convention pratique : les positions commencent à 1)



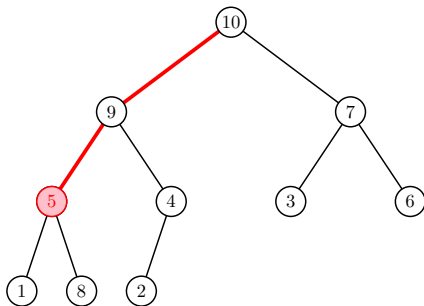
PRÉSERVER LA PROPRIÉTÉ DE TAS-MAX



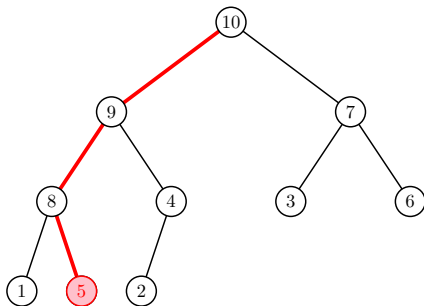
PRÉSERVER LA PROPRIÉTÉ DE TAS-MAX



PRÉSERVER LA PROPRIÉTÉ DE TAS-MAX



PRÉSERVER LA PROPRIÉTÉ DE TAS-MAX



PRÉSERVER LA PROPRIÉTÉ DE TAS-MAX

convention : les n éléments du tas sont rangés dans $T[1:n+1]$, et $T[0]$ contient la dernière position occupée $n = \text{taille}(T)$

Si les sous-arbres du nœud de position i sont des tas-max :

```
def entasser_max(T, i) :  
    max, l, r = i, gauche(i), droite(i)  
    if l <= taille(T) and T[l] > T[i] : max = l  
    if r <= taille(T) and T[r] > T[max] : max = r  
    if max != i :  
        T[i], T[max] = T[max], T[i]  
        entasser_max(T, max)
```

Complexité : $\Theta(h)$ si i est de hauteur h , donc $\Theta(\log n)$ au pire

TRANSFORMER UN TABLEAU EN TAS-MAX

remarque : les feuilles sont des tas-max (et il y en a $\lceil \frac{n}{2} \rceil$)

TRANSFORMER UN TABLEAU EN TAS-MAX

remarque : les feuilles sont des tas-max (et il y en a $\lceil \frac{n}{2} \rceil$)

```
def creer_tas_max(T) :  
    # parcours du tableau à l'envers  
    for i in range(taille(T)//2, 0, -1) :  
        entasser_max(T, i)
```

TRANSFORMER UN TABLEAU EN TAS-MAX

remarque : les feuilles sont des tas-max (et il y en a $\lceil \frac{n}{2} \rceil$)

```
def creer_tas_max(T) :  
    # parcours du tableau à l'envers  
    for i in range(taille(T)//2, 0, -1) :  
        entasser_max(T, i)
```

Théorème

si T est de taille n , *creer_tas_max*(T) transforme T en un tas-max en temps $\Theta(n)$ dans tous les cas

TRANSFORMER UN TABLEAU EN TAS-MAX

remarque : les feuilles sont des tas-max (et il y en a $\lceil \frac{n}{2} \rceil$)

```
def creer_tas_max(T) :  
    # parcours du tableau à l'envers  
    for i in range(taille(T)//2, 0, -1) :  
        entasser_max(T, i)
```

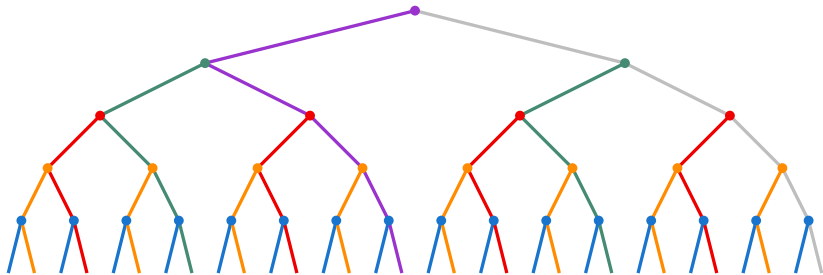
Théorème

si T est de taille n , *creer_tas_max*(T) transforme T en un tas-max en temps $\Theta(n)$ dans tous les cas

démonstration : *entasser_max*() est appelé une fois par nœud

\Rightarrow complexité totale en $\Theta \left(\sum_{v \text{ nœud de } T} h(v) \right)$

SOMME DES HAUTEURS DES NŒUDS



(code-couleur : une couleur par niveau, le nœud et le chemin correspondant sont colorés de la même couleur)

$$\Rightarrow \sum_{v \text{ nœud de } T} h(v) = \Theta(\text{nb d'arêtes}) = \Theta(n)$$

TRIER AVEC UN TAS-MAX

(toujours avec la convention que $T[0] = \text{taille}(T)$, mais facilement adaptable)

```
def tri_par_tas(T) :  
    creer_tas_max(T)  
    while taille(T) > 1 :  
        # mettre le max à sa place  
        T[1], T[taille(T)] = T[taille(T)], T[1]  
        # remettre le reste en tas  
        decrements_taille(T)  
        entasser_max(T, 1)  
    return T
```

Complexité : $\Theta(n \log n)$ au pire
(et non $\Theta(n)$, car c'est la somme des **profondeurs** cette fois!)

IMPLÉMENTER UNE FILE DE PRIORITÉ

structure destinée à gérer les priorités, par exemple pour l'ordonnancement de tâches sur un ordinateur

opérations supportées

- `insertion(F, x)`
- `maximum(F)`
- `extraction_max(F)`
- `augmenter_priorité(F, x, k)`

IMPLÉMENTER UNE FILE DE PRIORITÉ

structure destinée à gérer les priorités, par exemple pour l'ordonnancement de tâches sur un ordinateur

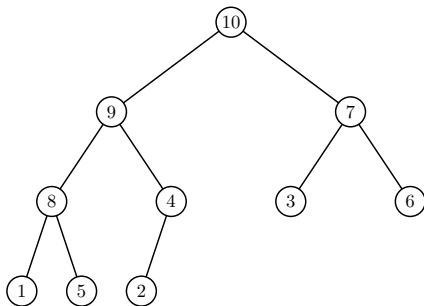
opérations supportées

- `insertion(F, x)`
- `maximum(F)`
- `extraction_max(F)`
- `augmenter_priorité(F, x, k)`

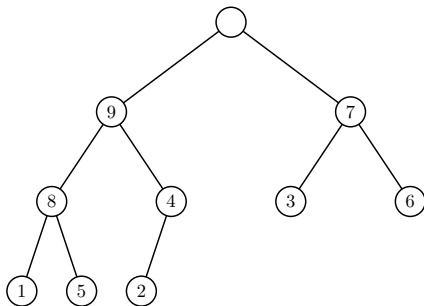
les tas-max sont particulièrement bien adaptés :

- recherche du maximum **en temps constant**
- les autres opérations se font en temps $\Theta(\log n)$

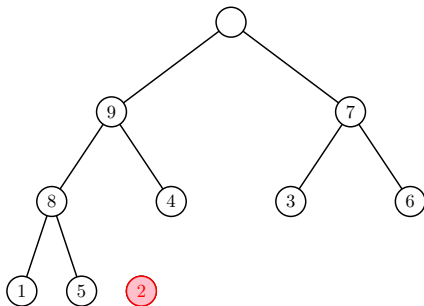
EXTRACTION DU MAXIMUM



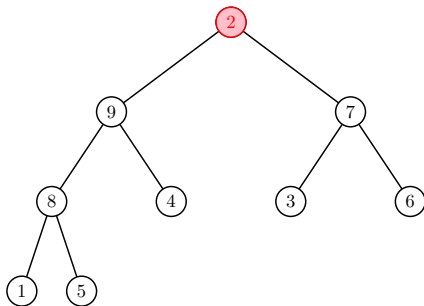
EXTRACTION DU MAXIMUM



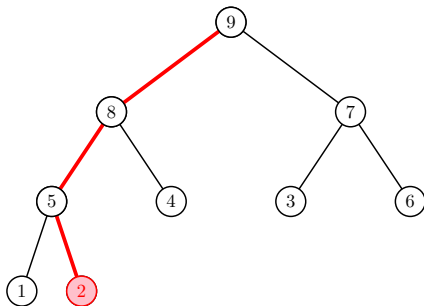
EXTRACTION DU MAXIMUM



EXTRACTION DU MAXIMUM



EXTRACTION DU MAXIMUM

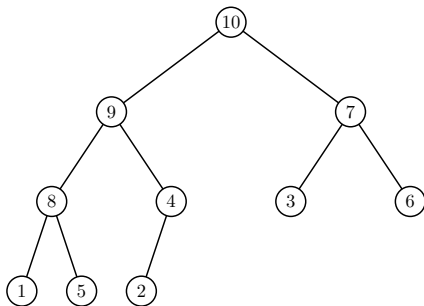


EXTRACTION DU MAXIMUM

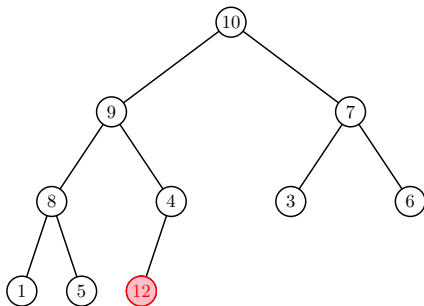
```
def extraction_maximum(F) :  
    max = F[1]  
    F[1] = F[taille(F)]  
    decrements_taille(F)  
    entasser_max(F, 1)  
    return max
```

Complexité : $\Theta(\log n)$ au pire

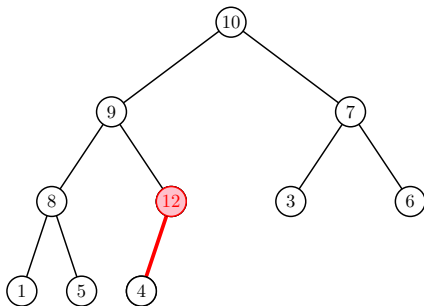
AUGMENTER UNE PRIORITÉ



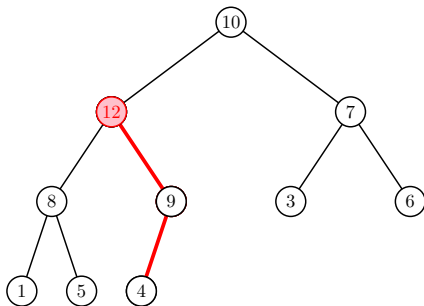
AUGMENTER UNE PRIORITÉ



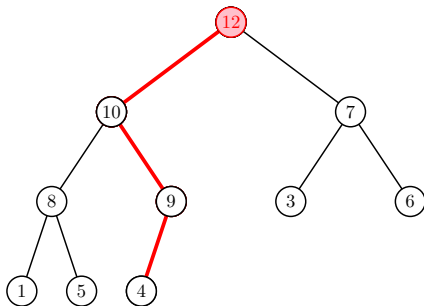
AUGMENTER UNE PRIORITÉ



AUGMENTER UNE PRIORITÉ



AUGMENTER UNE PRIORITÉ



MODIFIER UNE PRIORITÉ

```
def modifier_cle(F, i, cle) :  
    if cle < F[i] : # diminuer  
        F[i] = cle  
        entasser_max(F, i)  
    else : # augmenter  
        F[i] = cle  
        while i > 1 and F[pere(i)] < F[i] :  
            F[i], F[pere(i)] = F[pere(i)], F[i]  
            i = pere(i)
```

Complexité : $\Theta(\log n)$ au pire

INSÉRER UNE CLÉ

```
def inserer_cle(F, cle) :  
    augmenter_taille(F) # avec redimensionnement si nécessaire  
    F[taille(F)] = MIN # valeur inférieure à toutes les clés  
    modifier_cle(F, taille(F), cle)
```

Complexité :

- si le tableau est assez grand, $\Theta(\log n)$ au pire
- si un redimensionnement est nécessaire, $\Theta(n)$ — comme pour les tables de hachage, coût **amorti** constant si la taille est doublée à chaque fois

Une application des files de priorités
(et des dictionnaires) :
le codage de Huffman

COMPRESSION DE TEXTES

problème

étant donné un fichier `texte` de taille ℓ , construire un fichier `compresse(texte)` de taille strictement inférieure à ℓ tel que `texte` puisse être reconstruit à partir de `compresse(texte)`

COMPRESSION DE TEXTES

problème

étant donné un fichier **texte** de taille ℓ , construire un fichier **compresse**(**texte**) de taille strictement inférieure à ℓ tel que **texte** puisse être reconstruit à partir de **compresse**(**texte**)

(aparté)

il s'agit ici de **compression sans perte**, contrairement à ce qu'on a vu avec le **hachage**, ou, surtout, ce qu'on peut autoriser pour la compression de données **perceptibles** (image, son, video) pour laquelle on utilise le fait que l'oeil (ou l'oreille) ne peut pas capter tous les détails, et que le cerveau travaille

COMPRESSION DE TEXTES

problème

étant donné un fichier `texte` de taille ℓ , construire un fichier `compresse(texte)` de taille strictement inférieure à ℓ tel que `texte` puisse être reconstruit à partir de `compresse(texte)`

Malheureusement, un simple comptage montre :

Théorème

Aucun algorithme ne peut compresser tous les fichiers possibles.

COMPRESSION DE TEXTES

problème

étant donné un fichier `texte` de taille ℓ , construire un fichier `compresse(texte)` de taille strictement inférieure à ℓ tel que `texte` puisse être reconstruit à partir de `compresse(texte)`

Malheureusement, un simple comptage montre :

Théorème

Aucun algorithme ne peut compresser tous les fichiers possibles.

Alors ?

COMPRESSION DE TEXTES

de quoi part-on ?

d'un **texte** dont chaque caractère est codé en **ASCII** (sur un octet), ou éventuellement en **unicode** (4 octets pour l'**UTF32**)

COMPRESSION DE TEXTES

de quoi part-on ?

d'un **texte** dont chaque caractère est codé en **ASCII** (sur un octet), ou éventuellement en **unicode** (4 octets pour l'**UTF32**)

un octet par caractère pour un alphabet de 26 lettres ?

COMPRESSION DE TEXTES

de quoi part-on ?

d'un **texte** dont chaque caractère est codé en **ASCII** (sur un octet), ou éventuellement en **unicode** (4 octets pour l'**UTF32**)

un octet par caractère pour un alphabet de 26 lettres ?

- majuscules ou minuscules,

$$26 \times 2 = 52$$

COMPRESSION DE TEXTES

de quoi part-on ?

d'un **texte** dont chaque caractère est codé en **ASCII** (sur un octet), ou éventuellement en **unicode** (4 octets pour l'**UTF32**)

un octet par caractère pour un alphabet de 26 lettres ?

- majuscules ou minuscules, $26 \times 2 = 52$
- accentuées ou non, $> 12 \times 2 = 24$

COMPRESSION DE TEXTES

de quoi part-on ?

d'un **texte** dont chaque caractère est codé en **ASCII** (sur un octet), ou éventuellement en **unicode** (4 octets pour l'**UTF32**)

un octet par caractère pour un alphabet de 26 lettres ?

- majuscules ou minuscules, $26 \times 2 = 52$
- accentuées ou non, $> 12 \times 2 = 24$
- des caractères de ponctuation, espaces... > 10

COMPRESSION DE TEXTES

de quoi part-on ?

d'un **texte** dont chaque caractère est codé en **ASCII** (sur un octet), ou éventuellement en **unicode** (4 octets pour l'**UTF32**)

un octet par caractère pour un alphabet de 26 lettres ?

- majuscules ou minuscules, $26 \times 2 = 52$
- accentuées ou non, $> 12 \times 2 = 24$
- des caractères de ponctuation, espaces... > 10
- des chiffres 10

COMPRESSION DE TEXTES

de quoi part-on ?

d'un **texte** dont chaque caractère est codé en **ASCII** (sur un octet), ou éventuellement en **unicode** (4 octets pour l'**UTF32**)

un octet par caractère pour un alphabet de 26 lettres ?

- majuscules ou minuscules, $26 \times 2 = 52$
- accentuées ou non, $> 12 \times 2 = 24$
- des caractères de ponctuation, espaces... > 10
- des chiffres 10

soit une centaine de caractères, nécessitant donc 7 bits chacun
si chaque caractère est codé avec le même nombre de bits

CODE DE LONGUEUR VARIABLE

constat

tous ces caractères n'ont pas du tout la même fréquence dans un texte écrit en français, par exemple

CODE DE LONGUEUR VARIABLE

constat

tous ces caractères n'ont pas du tout la même fréquence dans un texte écrit en français, par exemple

corollaire

la probabilité uniforme sur les textes de longueur donnée n'est pas pertinente, et il est peut-être possible de compresser les textes « réels » (au détriment des « textes » n'ayant aucun sens)

CODE DE LONGUEUR VARIABLE

constat

tous ces caractères n'ont pas du tout la même fréquence dans un texte écrit en français, par exemple

corollaire

la probabilité uniforme sur les textes de longueur donnée n'est pas pertinente, et il est peut-être possible de compresser les textes « réels » (au détriment des « textes » n'ayant aucun sens)

idée

donner des **mots de code** de longueur *variable* aux lettres en fonction de leur *fréquence* dans le texte considéré : (très) courts pour les lettres fréquentes, plus longs pour les lettres rares (et ignorer les caractères qui n'apparaissent pas)

CODAGE...

- utiliser une *table de hachage* pour stocker les mots de code des différents caractères
- coder le texte par simple *concaténation* des codes des caractères qui le composent

CODAGE...

- utiliser une *table de hachage* pour stocker les mots de code des différents caractères
- coder le texte par simple *concaténation* des codes des caractères qui le composent

en ASCII

```
>>> texte = 'abracadabra'
>>> ascii = { 'a' : '01100001', 'b' : '01100010',
              'c' : '01100011', 'd' : '01100100', 'r' : '01110010' }
>>> ''.join(ascii[c] for c in texte)
'01100001011000100111001001100001011000110110000101100
10001100001011000100111001001100001'
>>> len(''.join(ascii[c] for c in texte))
```

avec un code « mini-ASCII »

```
>>> texte = 'abracadabra'                                # longueur 11
>>> dico = { 'a' : '001', 'b' : '010', 'c' : '011',
              'd' : '100', 'r' : '101' }
>>> ''.join(dico[c] for c in texte)
'001010101001011001100001010101001'                    # longueur 33
```

avec un code « mini-ASCII »

```
>>> texte = 'abracadabra'                                # longueur 11
>>> dico = { 'a' : '001', 'b' : '010', 'c' : '011',
              'd' : '100', 'r' : '101' }
>>> ''.join(dico[c] for c in texte)
'001010101001011001100001010101001'                    # longueur 33
```

avec un code de longueur variable

```
>>> huffman = { 'a' : '0', 'b' : '10', 'r' : '110',
                 'c' : '1110', 'd' : '1111' }
>>> ''.join(huffman[c] for c in texte)
'01011001110011110101100'                                # longueur 23
```

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

```
texte_encode = '001010101001011001100001010101001'
```

... ET DÉCODAGE

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

```
texte_encode = '001010101001011001100001010101001'
```

```
texte_decode = 'a'
```

... ET DÉCODAGE

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

```
texte_encode = '001010101001011001100001010101001'
```

```
texte_decode = 'ab'
```

... ET DÉCODAGE

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

```
texte_encode = '001010101001001011001100001010101001'
```

```
texte_decode = 'abr'
```

... ET DÉCODAGE

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

```
texte_encode = '001010101001011001100001010101001'
```

```
texte_decode = 'abra
```


... ET DÉCODAGE

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

```
texte_encode = '001010101001011001100001010101001'
```

```
texte_decode = 'abrac'
```

... ET DÉCODAGE

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

```
texte_encode = '001010101001011001100001010101001'
```

```
texte_decode = 'abraca'
```

... ET DÉCODAGE

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

```
texte_encode = '001010101001011001100001010101001'
```

```
texte_decode = 'abraca
```

... ET DÉCODAGE

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

```
texte_encode = '001010101001011001100001010101001'
```

```
texte_decode = 'abracada'
```

... ET DÉCODAGE

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

```
texte_encode = '001010101001011001100001010101001'
```

```
texte_decode = 'abracadab'
```

... ET DÉCODAGE

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

```
texte_encode = '001010101001011001100001010101001'
```

```
texte_decode = 'abracadabr'
```

... ET DÉCODAGE

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

```
texte_encode = '001010101001011001100001010101001'
```

```
texte_decode = 'abracadabra'
```

avec un code « mini-ASCII », c'est facile

```
>>> code = ''.join(dico[c] for c in texte)
>>> decoupe = [ code[i:i+3] for i in range(0,33,3) ]
['001', '010', '101', '001', '011', '001', '100',
'001', '010', '101', '001']
>>> inverse = { v : k for (k,v) in dico.items() }
{ '011': 'c', '010': 'b', '100': 'd', '001': 'a',
'101': 'r' }
>>> ''.join(inverse[elt] for elt in decoupe)
'abracadabra'
```


... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

```
texte_encode = '01011001110011110101100'
```

... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

```
texte_encode = '01011001110011110101100'
```

```
texte_decode = 'a'
```

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

```
texte_encode = '0 1011001110011110101100'
```

```
texte_decode = 'a
```

... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

```
texte_encode = '0 1011001110011110101100'
```

```
texte_decode = 'ab'
```

... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

```
texte_encode = '0 10 11001110011110101100'
```

```
texte_decode = 'ab'
```

... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

```
texte_encode = '0 10 11001110011110101100'
```

```
texte_decode = 'ab'
```

... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

```
texte_encode = '0 10 11001110011110101100'
```

```
texte_decode = 'abr'
```

... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

```
texte_encode = '0 10 110 01110011110101100'
```

```
texte_decode = 'abra
```


... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

```
texte_encode = '0 10 110 0 1110011110101100'
```

```
texte_decode = 'abra
```

... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

```
texte_encode = '0 10 110 0 1110011110101100'
```

```
texte_decode = 'abra
```

... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

```
texte_encode = '0 10 110 0 1110011110101100'
```

```
texte_decode = 'abra
```

... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

```
texte_encode = '0 10 110 0 1110011110101100'
```

```
texte_decode = 'abrac'
```

... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110 011110101100'

`texte_decode` = 'abraca

... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110 0 11110101100'

`texte_decode` = 'abraca'

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

```
texte_encode = '0 10 110 0 1110 0 11110101100'
```

```
texte_decode = 'abraca'
```

... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

```
texte_encode = '0 10 110 0 1110 0 11110101100'
```

```
texte_decode = 'abraca'
```


... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110 0 11110101100'

`texte_decode` = 'abraca**d**'

... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110 0 1111 0101100'

`texte_decode` = 'abracada

... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

```
texte_encode = '0 10 110 0 1110 0 1111 0 101100'
```

```
texte_decode = 'abracada'
```

... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110 0 1111 0 101100'

`texte_decode` = 'abracadab'

... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110 0 1111 0 10 1100'

`texte_decode` = 'abracadab'

... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110 0 1111 0 10 1100'

`texte_decode` = 'abracadab'

... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110 0 1111 0 10 1100'

`texte_decode` = 'abracadabr'

... ET DÉCODAGE

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

```
texte_encode = '0 10 110 0 1110 0 1111 0 10 110 0'
```

```
texte_decode = 'abracadabra'
```


CODE PRÉFIXE

pourquoi avons-nous réussi ?

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

CODE PRÉFIXE

pourquoi avons-nous réussi ?

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

car aucun mot de code n'est **préfixe** d'un autre mot de code

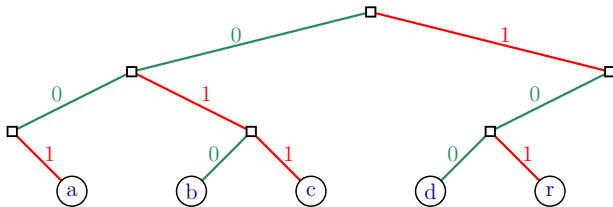
c'est la définition d'un **code préfixe**

CODE PRÉFIXE

autre formulation

les mots du code sont les (codes des) **feuilles** d'un arbre binaire

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

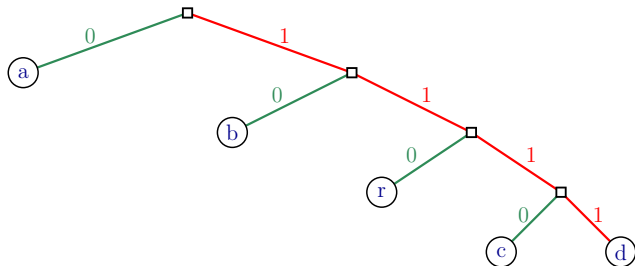


CODE PRÉFIXE

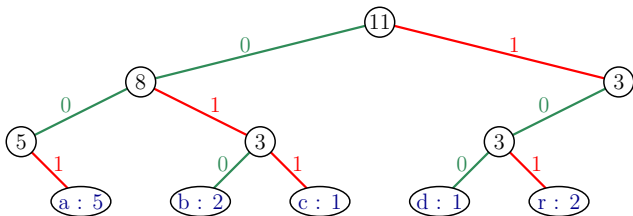
autre formulation

les mots du code sont les (codes des) **feuilles** d'un arbre binaire

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110



COMPARAISON DE TAUX DE COMPRESSION



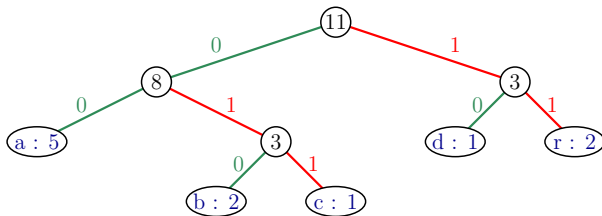
pour chaque caractère c , soit

- $n(c)$ le nombre d'occurrences de c dans le texte
- $p(c)$ la longueur du code de c (*i.e.* sa profondeur dans l'arbre)

longueur totale du texte codé :

$$\sum_c n(c)p(c) = 33$$

COMPARAISON DE TAUX DE COMPRESSION



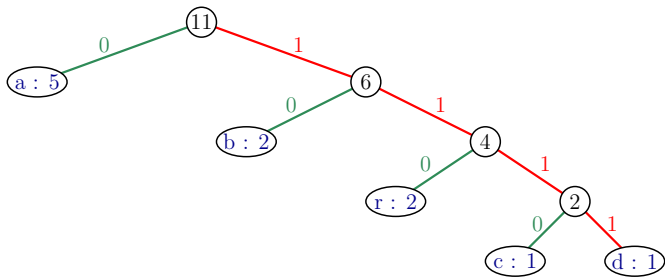
pour chaque caractère c , soit

- $n(c)$ le nombre d'occurrences de c dans le texte
- $p(c)$ la longueur du code de c (*i.e.* sa profondeur dans l'arbre)

longueur totale du texte codé :

$$\sum_c n(c)p(c) = 25$$

COMPARAISON DE TAUX DE COMPRESSION



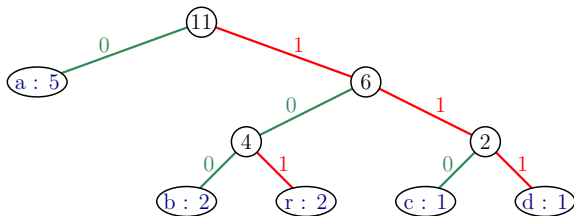
pour chaque caractère c , soit

- $n(c)$ le nombre d'occurrences de c dans le texte
- $p(c)$ la longueur du code de c (*i.e.* sa profondeur dans l'arbre)

longueur totale du texte codé :

$$\sum_c n(c)p(c) = 23$$

COMPARAISON DE TAUX DE COMPRESSION



pour chaque caractère c , soit

- $n(c)$ le nombre d'occurrences de c dans le texte
- $p(c)$ la longueur du code de c (*i.e.* sa profondeur dans l'arbre)

longueur totale du texte codé :

$$\sum_c n(c)p(c) = 23$$

COMMENT CES DEUX DERNIERS CODES SONT-ILS OBTENUS ?

codage de Huffman

construire l'arbre de manière *gloutonne* :

- placer les couples (caractère, fréquence) dans une file de priorité
- tant que la file de priorité contient plusieurs éléments
 - extraire les deux éléments de fréquence minimale
 - créer un nœud binaire dont ces éléments sont les fils
 - insérer le nœud dans la file de priorité
- retourner l'unique élément de la file de priorité – la racine de l'arbre de code

DÉROULEMENT DE L'ALGORITHME

File :

a : 5

b : 2

r : 2

c : 1

d : 1

DÉROULEMENT DE L'ALGORITHME

File :

a : 5

b : 2

r : 2

c : 1

d : 1

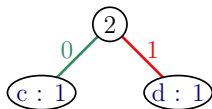
DÉROULEMENT DE L'ALGORITHME

File :

a : 5

b : 2

r : 2



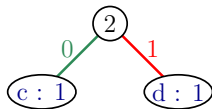
DÉROULEMENT DE L'ALGORITHME

File :

a : 5

b : 2

r : 2



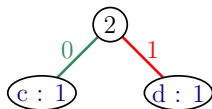
DÉROULEMENT DE L'ALGORITHME

File :

a : 5

b : 2

r : 2

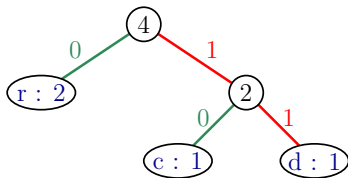


DÉROULEMENT DE L'ALGORITHME

File :

a : 5

b : 2

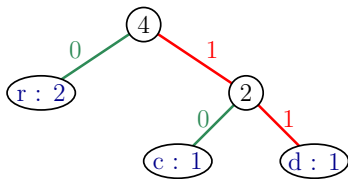


DÉROULEMENT DE L'ALGORITHME

File :

a : 5

b : 2

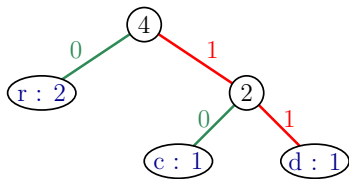


DÉROULEMENT DE L'ALGORITHME

File :

a : 5

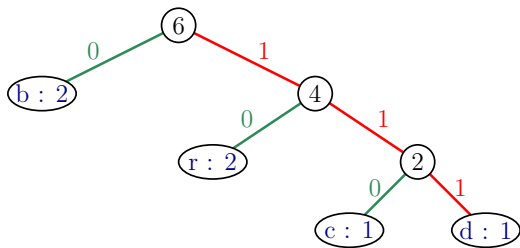
b : 2



DÉROULEMENT DE L'ALGORITHME

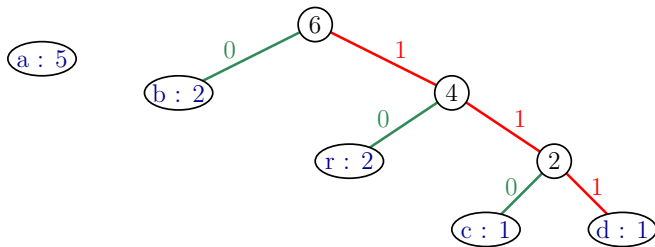
File :

a : 5

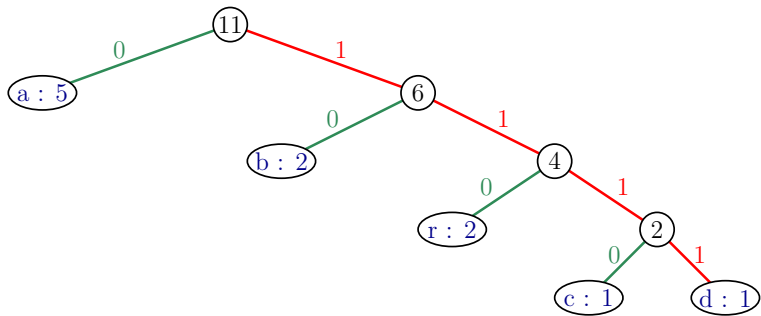


DÉROULEMENT DE L'ALGORITHME

File :



DÉROULEMENT DE L'ALGORITHME



Théorème

le code produit par l'algorithme de Huffman est optimal parmi les codes préfixes

Théorème

le code produit par l'algorithme de Huffman est optimal parmi les codes préfixes

Lemme

soit x et y deux caractères de fréquence minimale, alors il existe un code préfixe optimal φ tel que $\varphi(x) = w \cdot 0$ et $\varphi(y) = w \cdot 1$ pour un même mot w .

Théorème

le code produit par l'algorithme de Huffman est optimal parmi les codes préfixes

Lemme

soit x et y deux caractères de fréquence minimale, alors il existe un code préfixe optimal tel que x et y ont le même père dans l'arbre associé.

Théorème

le code produit par l'algorithme de Huffman est optimal parmi les codes préfixes

Lemme

soit x et y deux caractères de fréquence minimale, alors il existe un code préfixe optimal tel que x et y ont le même père dans l'arbre associé.

Lemme

soit x et y deux caractères ayant le même père dans un arbre de codage T ; soit T' obtenu à partir de T en supprimant x et y , et en associant un caractère z à la nouvelle feuille. Alors T est optimal si et seulement si T' l'est.

Démonstration du 2^e lemme

*Soit t le texte initial, et t' le texte obtenu à partir de t en remplaçant les x et y par z . Le code de z ayant un bit de moins que ceux de x et de y , la longueur du codage de t par T est exactement égale à celle du codage de t' par T' **plus** le nombre d'occurrences de z , qui dépend seulement du texte t et pas de T . Donc ces longueurs sont minimales simultanément.*

POINTS NÉGATIFS

- on n'a pas compté le coût de la transmission du dictionnaire
- l'algorithme nécessite un précalcul des fréquences, donc une prélecture du texte, ce qui n'est pas toujours possible

POINTS NÉGATIFS

- on n'a pas compté le coût de la transmission du dictionnaire
 - l'algorithme nécessite un précalcul des fréquences, donc une prélecture du texte, ce qui n'est pas toujours possible
-
- si la langue du texte est connue, on peut utiliser une version *statique* avec un dictionnaire prédéfini... mais on perd nécessairement l'optimalité

POINTS NÉGATIFS

- on n'a pas compté le coût de la transmission du dictionnaire
 - l'algorithme nécessite un précalcul des fréquences, donc une prélecture du texte, ce qui n'est pas toujours possible
-
- si la langue du texte est connue, on peut utiliser une version *statique* avec un dictionnaire prédéfini... mais on perd nécessairement l'optimalité
 - il existe une version *adaptative* qui recalcule les fréquences au fur et à mesure de la lecture (et du codage) ; elle compresse encore mieux, *mais* la complexité en temps est bien plus grande puisque l'arbre doit être recalculé à chaque caractère