Introduction à l'algorithmique : Tris

Serge Haddad

LMF, ENS Paris-Saclay & CNRS & Inria

Agrégation Informatique

1. Tris par comparaison
2. Borne inférieure de complexité
3. Tris sans comparaison
4. Applications

Cadre général

En entrée :

- Un type tp muni d'une relation d'ordre total
- Un tableau de n valeurs typées par tp.

En sortie le tableau contient :

- une permutation des valeurs du tableau
- triées selon l'ordre de tp.

Quelques hypothèses supplémentaires.

- tp peut être infini ou fini mais dépendant de paramètres.
- tp peut être structuré.

Objectifs.

- Complexité temporelle et spatiale
- Stabilité
- Adaptation au *désordre* initial

Plan

1. Tris par comparaison

Borne inférieure de complexité

Tris sans comparaison

Applications

Tri par insertion

« Insérer » en bonne position $T[i]$ dans $T[1, i-1]$ trié
de telle sorte que $T[1, i]$ soit trié.

Tri(T, n)

For i **from** 2 **to** n **do**

$temp \leftarrow T[i]$

$j \leftarrow i - 1$

While $j > 0$ **and** $T[j] > temp$ **do** $T[j+1] \leftarrow T[j]$; $j \leftarrow j - 1$

$T[j+1] \leftarrow temp$

Analyse du tri par insertion

Le tri par insertion est un tri stable.

Le tri par insertion est un tri *en place*.

Sa complexité est en $\Theta(n^2)$.

C'est un tri adaptatif.

Soit :

$$\text{Disordered} = \{(i, j) \mid i < j \wedge T[i] > T[j]\}$$

Alors sa complexité est en $\Theta(n + |\text{Disordered}|)$.

Tri par insertion efficace

Soit A un arbre binaire de recherche équilibré *avec occurrences multiples*.

Tri(T, n)

$A \leftarrow \emptyset$

For i **from** 1 **to** n **do** Insérer$(A, T[i])$

$Ind \leftarrow 1$

Copier$(A.root)$

Copier$(vertex)$

If $vertex.fg \neq$ `NULL` **then** Copier$(vertex.fg)$

$T[ind] \leftarrow vertex.value$; $ind \leftarrow ind + 1$

If $vertex.fd \neq$ `NULL` **then** Copier$(vertex.fd)$

D'où une complexité en $\Theta(n \log(n))$.

Tri fusion

- Diviser le tableau en deux parties approximativement égales.
- Trier les sous-tableaux.
- Fusionner les sous-tableaux triés.

Tri(T, n)
If $n = 1$ **then return** ;
$mid \leftarrow \lfloor \frac{n}{2} \rfloor$;
For i **from** 1 **to** mid **do** $T_1[i] \leftarrow T[i]$;
For i **from** $mid + 1$ **to** n **do** $T_2[i - mid] \leftarrow T[i]$;
Tri(T_1, mid) ;
Tri$(T_2, n - mid)$;
Fusion$(T_1, mid, T_2, n - mid, T)$;

Fusion

- Maintenir un indice par tableau
- Recopier la plus petite valeur et incrémenter l'indice correspondant
- Recopier la partie du dernier tableau restant

Fusion(T_1, n_1, T_2, n_2, T)

$i \leftarrow 1$; $i_1 \leftarrow 1$; $i_2 \leftarrow 1$;

While $i_1 \leq n_1$ **and** $i_2 \leq n_2$ **do**

  If $T_1[i_1] \leq T_2[i_2]$ **then** $T[i] \leftarrow T_1[i_1]$; $i_1 \leftarrow i_1 + 1$;

  Else $T[i] \leftarrow T_2[i_2]$; $i_2 \leftarrow i_2 + 1$;

  $i \leftarrow i + 1$;

For j **from** i_1 **to** n_1 **do** $T[i] \leftarrow T_1[j]$; $i \leftarrow i + 1$;

For j **from** i_2 **to** n_2 **do** $T[i] \leftarrow T_2[j]$; $i \leftarrow i + 1$;

$t_{\text{Fusion}}(n_1, n_2) = \Theta(n_1 + n_2)$ $\qquad$ $t_{\text{Tri}}(n) = t_{\text{Tri}}(\lfloor \frac{n}{2} \rfloor) + t_{\text{Tri}}(\lceil \frac{n}{2} \rceil) + \Theta(n)$

D'où :

$$\boxed{t_{\text{Tri}}(n) = \Theta(n \log(n))}$$

Analyse du tri fusion

Le tri fusion est un tri stable.

Son temps de calcul est indépendant des données (pas adaptatif).

Le tri fusion n'est pas un tri *en place*.

Les tableaux auxiliaires occupent n cellules.

En cumulant la place occupée par les appels récursifs, on obtient :

$$n + \frac{n}{2} + \frac{n}{4} + \ldots \sim 2n$$

On peut se limiter à un unique tableau auxiliaire
en alternant les copies d'un tableau à l'autre.

Tri fusion partiel

Toutes les modifications du tableau se font par échange.

Swap(i, j) $temp \leftarrow T[i]$; $T[i] \leftarrow T[j]$; $T[j] \leftarrow temp$

Le tri fusion partiel ordonne un *bloc* du tableau
en utilisant un bloc disjoint de même taille comme zone de travail

TriP(a, b, m) $\{[a, a + m[\cap[b, b + m[= \emptyset\}$
If $m = 1$ **then return** ;
$mid \leftarrow \lfloor \frac{m}{2} \rfloor$;
For i **from** 0 **to** $m - 1$ **do** Swap$(a + i, b + i)$;
TriP(b, a, mid) ;
TriP$(b + mid, a + mid, m - mid)$;
FusionP$(b, mid, b + mid, m - mid, a)$;

Fusion partielle

Identique à la fusion mais procède par échange.

$\{[a, a+n_1[\cap[b, b+n_2[= [a, a+n_1[\cap[c, c+n_1+n_2[= [c, c+n_1+n_2[\cap[b, b+n_2[= \emptyset\}$

FusionP(a, n_1, b, n_2, c)

$i \leftarrow c$; $i_1 \leftarrow a$; $i_2 \leftarrow b$;

While $i_1 < a + n_1$ **and** $i_2 < b + n_2$ **do**

  If $T[i_1] \leq T[i_2]$ **then** Swap(i, i_1) ; $i_1 \leftarrow i_1 + 1$;

  Else Swap(i, i_2) ; $i_2 \leftarrow i_2 + 1$;

  $i \leftarrow i + 1$;

For j **from** i_1 **to** $a + n_1 - 1$ **do** Swap(i, j) ; $i \leftarrow i + 1$;

For j **from** i_2 **to** $b + n_2 - 1$ **do** Swap(i, j) ; $i \leftarrow i + 1$;

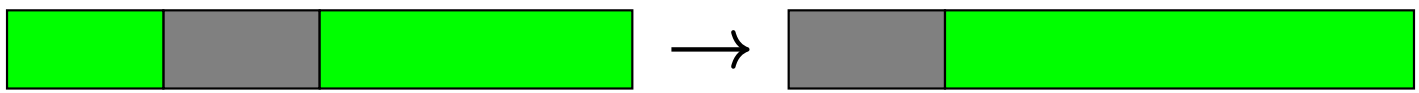
Comme précédemment :

$$t_{\mathsf{TriP}}(m) = \Theta(m \log(m))$$

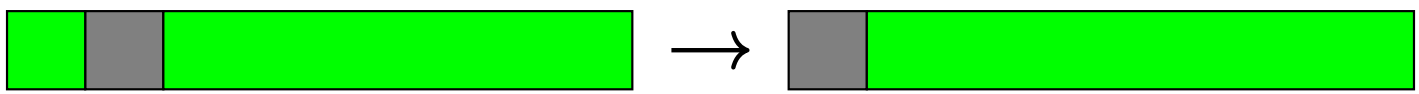
Tri fusion sur place (1)

• Le tri fusion sur place trie la moitié droite du tableau
où la zone de travail est la moitié gauche.

• Puis il trie le quart gauche du tableau et le fusionne avec la moitié droite
où la zone de travail est le deuxième quart.



• Puis il trie le huitième gauche du tableau et le fusionne avec les trois quarts droits
où la zone de travail est le deuxième huitième . . .



• Lorsqu'il ne reste que le premier élément à trier,
il est inséré en bonne position.

Tri fusion sur place (2)

Tri()

$mid \leftarrow \lfloor \frac{n}{2} \rfloor$;

TriP$(n - mid + 1, n - 2 * mid + 1, mid)$;

$m \leftarrow n - mid$;

While $m > 1$ **do**

  $mid \leftarrow \lfloor \frac{m}{2} \rfloor$;

  TriP$(1, mid, mid)$;

  Fusion$(mid, n - m)$;

  $m \leftarrow m - mid$;

$i \leftarrow 2$;

While $i \leq n$ **and** $T[i] < T[i-1]$ **do** Swap$(i, i-1)$; $i \leftarrow i + 1$;

Fusion (presque) sur place

$\{2 * n_1 + n_2 \leq n\}$

Fusion(n_1, n_2)

$i \leftarrow n - n_1 - n_2 + 1$; $i_1 \leftarrow 1$; $i_2 \leftarrow n - n_2 + 1$;

While $i_1 \leq n_1$ **and** $i_2 \leq n$ **do**

  If $T[i_1] < T[i_2]$ **then** Swap(i, i_1) ; $i_1 \leftarrow i_1 + 1$;

  Else Swap(i, i_2) ; $i_2 \leftarrow i_2 + 1$;

  $i \leftarrow i + 1$;

For j **from** i_1 **to** n_1 **do** Swap(i, j) ; $i \leftarrow i + 1$;

Invariant de boucle.

- Les blocs bleus, verts et jaunes sont triés.
- Les items « jaunes » sont plus petits que les « bleus » et les « verts ».
- Les tailles des blocs bleus et marrons sont égales.

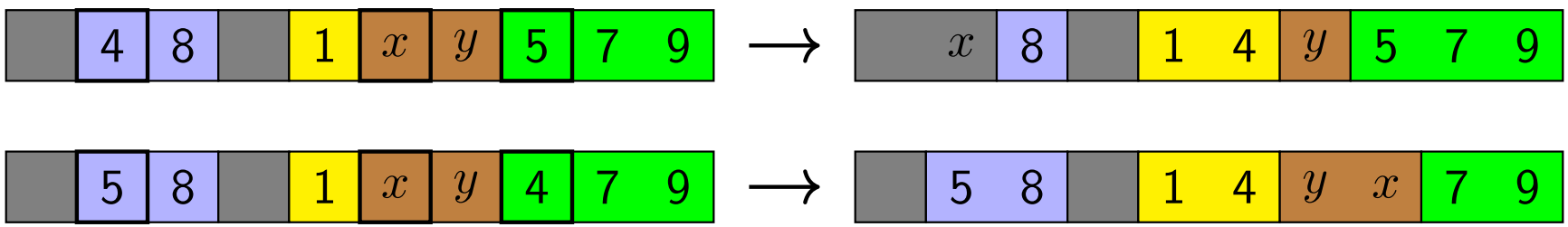
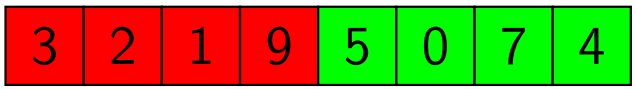
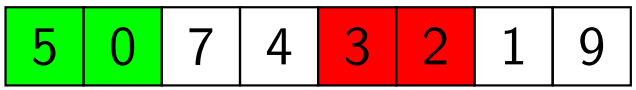


Illustration (1)

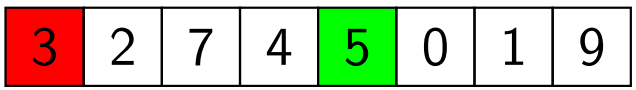
TriP$(5, 1, 4)$



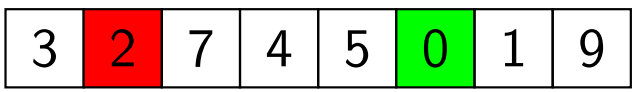
TriP$(1, 5, 2)$



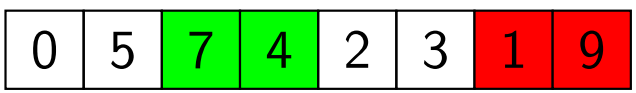
TriP$(5, 1, 1)$



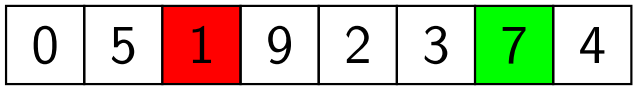
TriP$(6, 2, 1)$



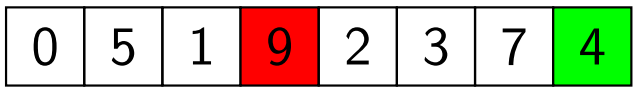
TriP$(3, 7, 2)$



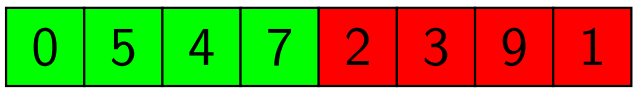
TriP$(7, 3, 1)$



TriP$(8, 4, 1)$



FusionP$(1, 2, 3, 2, 5)$



Après TriP$(5, 1, 4)$

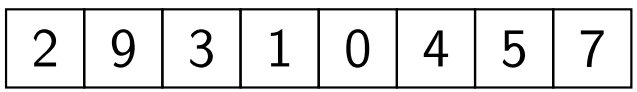
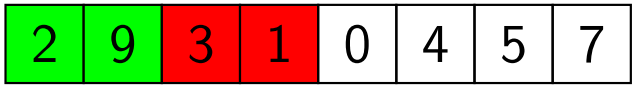
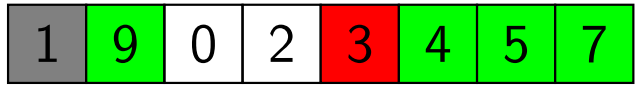


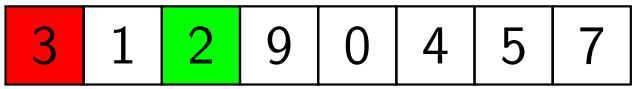
Illustration (2)

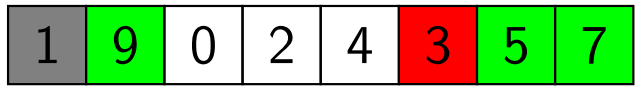
TriP$(1, 3, 2)$



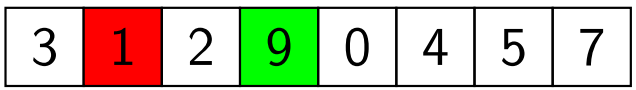


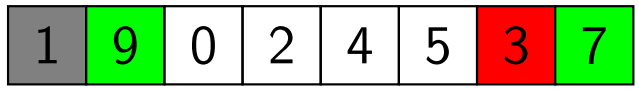
TriP$(3, 1, 1)$



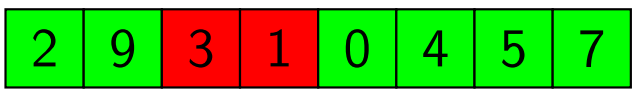


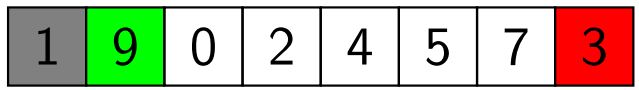
TriP$(4, 2, 1)$

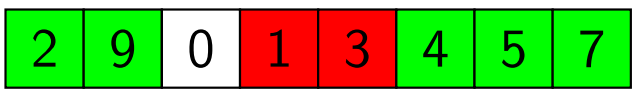




Fusion$(2, 4)$







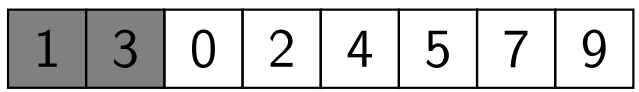
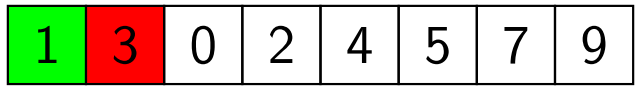
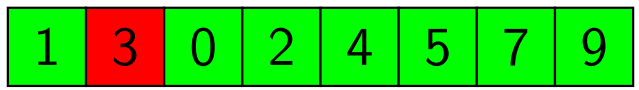


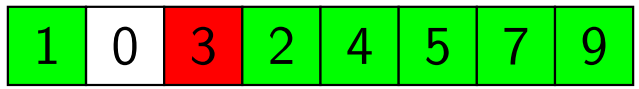
Illustration (3)

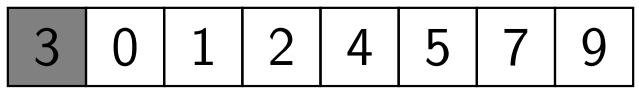
TriP$(1, 2, 1)$



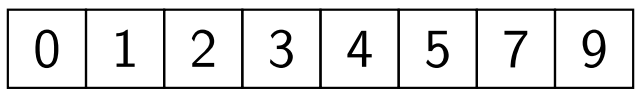
Fusion$(1, 6)$







Insertion du premier élément



Complexité du tri fusion sur place

Soit $t_{\mathsf{TriP}}(m) \leq cm\log(m)$.

Supposons sans perte de généralité $n = 2^k$.

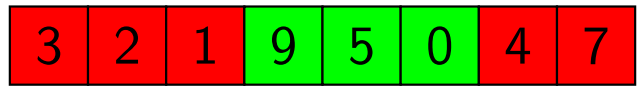
$$
\begin{aligned}
t_{\mathsf{Tri}}(n) \leq & \sum_{i=1}^{k} t_{\mathsf{TriP}}(\frac{n}{2^i}) + \sum_{i=1}^{k-1} dn + en \\
\leq & \sum_{i=1}^{k} c\frac{n}{2^i}\log(n) + d(k-1)n + en \\
\leq & (c+d+e)kn \\
= & (c+d+e)n\log(n)
\end{aligned}
$$

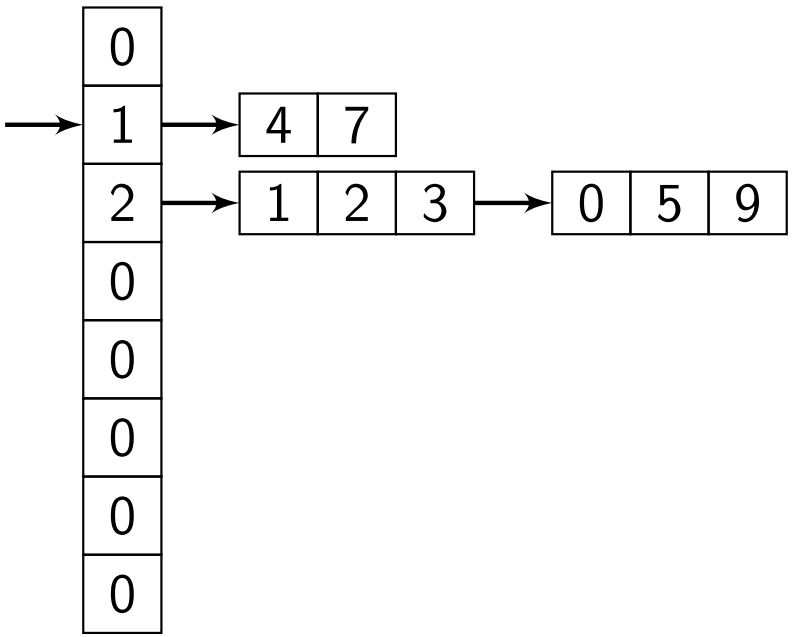
d, constante de Fusion

e, constante de l'insertion du dernier élément

Tri fusion adaptatif (1)

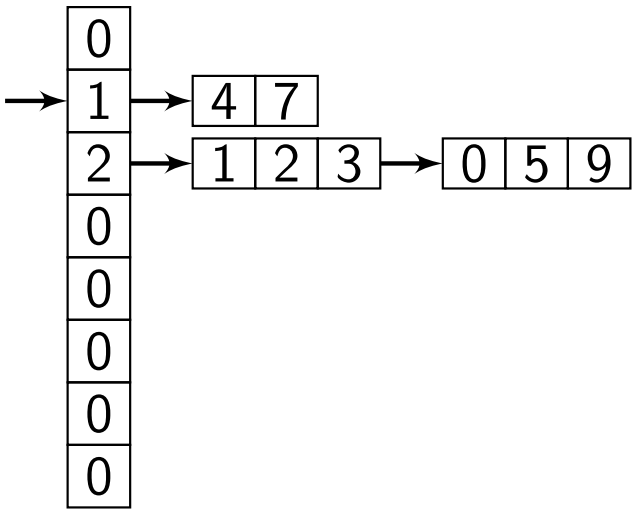
Construction de séquences ordonnées

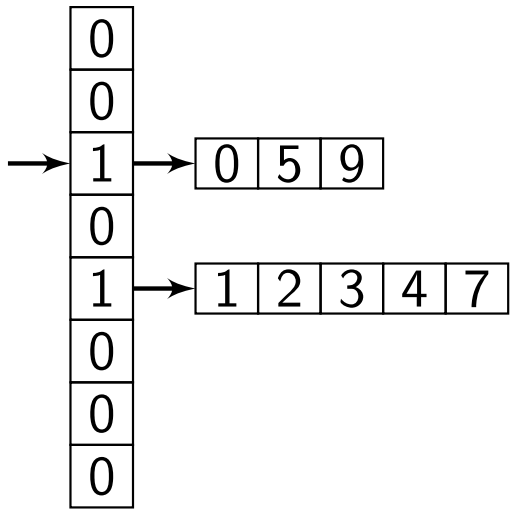


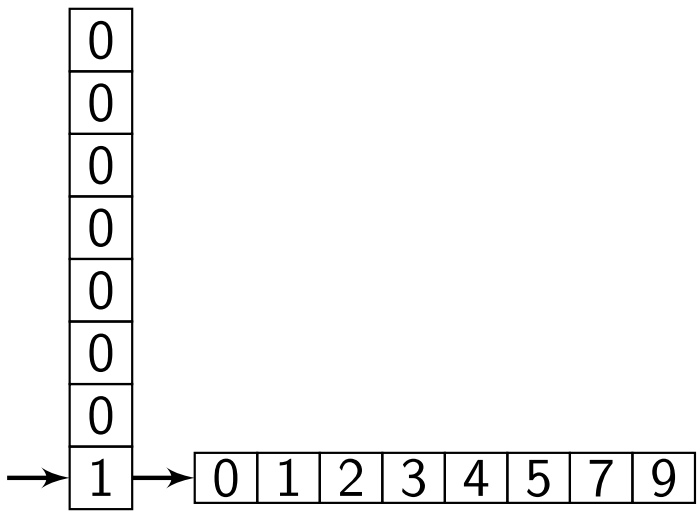


Tri fusion adaptatif (2)

Fusion de séquences ordonnées







Construction de séquences ordonnées

For i **from** 1 **to** n **do** $TL[i] \leftarrow$ CreerListeListe() ; $Tnb[i] \leftarrow 0$

$Liste \leftarrow$ CreerListeInt$(T[1])$; $\ell \leftarrow 1$; $new \leftarrow \top$; $B \leftarrow 1$

For i **from** 2 **to** n **do**

  If new **then** $new \leftarrow \bot$; $up \leftarrow T[i-1] \leq T[i]$

  If up **and** $T[i-1] \leq T[i]$ **then** $Liste \leftarrow Liste \cdot T[i]$; $\ell \leftarrow \ell + 1$

  Else If not up **and** $T[i-1] \geq T[i]$ **then** $Liste \leftarrow T[i] \cdot Liste$; $\ell \leftarrow \ell + 1$

  Else

    $TL[\ell] \leftarrow TL[\ell] \cdot Liste$; $Tnb[\ell] \leftarrow Tnb[\ell] + 1$

    $Liste \leftarrow$ CreerListeInt$(T[i])$; $\ell \leftarrow 1$; $new \leftarrow \top$; $B \leftarrow B + 1$

$TL[\ell] \leftarrow TL[\ell] \cdot Liste$; $Tnb[\ell] \leftarrow Tnb[\ell] + 1$

Observation. La construction opère en temps linéaire.

Fusion de séquences ordonnées

$ind \leftarrow 1$

For i **from** B **downto** 2 **do**

  While $Tnb[ind] = 0$ **do** $ind \leftarrow ind + 1$

  $L1 \leftarrow$ Extraire$(TL[ind])$; $Tnb[ind] \leftarrow Tnb[ind] - 1$; $\ell \leftarrow ind$

  While $Tnb[ind] = 0$ **do** $ind \leftarrow ind + 1$

  $L2 \leftarrow$ Extraire$(TL[ind])$; $Tnb[ind] \leftarrow Tnb[ind] - 1$; $\ell \leftarrow \ell + ind$

  $L \leftarrow$ Fusion$(L1, L2)$; $TL[\ell] \leftarrow L \cdot TL[\ell]$; $Tnb[\ell] \leftarrow Tnb[\ell] + 1$

$L \leftarrow$ Extraire$(TL[n])$

For i **from** 1 **to** n **do** $T[i] \leftarrow$ Extraire(L)

Analyse de complexité (1)

Soit $n_1 \leq n_2 \ldots \leq n_i$, la taille des i listes rangées dans TL au début d'une itération de fusion.

Soit $\Phi(n_1, n_2, \ldots, n_i) = \sum_{j \leq i} n_j \log_{\frac{3}{2}}(\frac{n}{n_j})$.

$$\begin{aligned}(n_1 + n_2) \log_{\frac{3}{2}}(\frac{n}{n_1 + n_2}) = & \quad n_1 \log_{\frac{3}{2}}(\frac{n}{n_1 + n_2}) + n_2 \log_{\frac{3}{2}}(\frac{n}{n_1 + n_2}) \\ \leq & \quad n_1 \log_{\frac{3}{2}}(\frac{n}{n_1}) + n_2 \log_{\frac{3}{2}}(\frac{n}{n_2})\end{aligned}$$

D'où :

$$\boxed{\Phi(n_1 + n_2, \ldots, n_i) \leq \Phi(n_1, n_2, \ldots, n_i)}$$

Analyse de complexité (2)

Si $L2$ est obtenue par fusion de $L1'$ et $L2'$,

- Soit $|L1| = |L2|$;
- Soit $|L1| < |L2|$, $L1$ était présent au moment de la fusion ce qui implique $|L1| \geq \max(|L1'|, |L2'|)$.

D'où :

$$\boxed{2|L1| \geq |L2|}$$

$$\begin{aligned}
(n_1 + n_2)\log_{\frac{3}{2}}(\frac{n}{n_1+n_2}) = & \quad n_1 \log_{\frac{3}{2}}(\frac{n}{n_1+n_2}) + n_2 \log_{\frac{3}{2}}(\frac{n}{n_1+n_2}) \\
\leq & \quad n_1 \log_{\frac{3}{2}}(\frac{2n}{3n_2}) + n_2 \log_{\frac{3}{2}}(\frac{2n}{3n_2}) \\
\leq & \quad n_1(\log_{\frac{3}{2}}(\frac{n}{n_1}) - 1) + n_2(\log_{\frac{3}{2}}(\frac{n}{n_2} - 1)
\end{aligned}$$

D'où :

$$\boxed{\Phi(n_1 + n_2, \ldots, n_i) \leq \Phi(n_1, n_2, \ldots, n_i) - (n_1 + n_2)}$$

Observation. Φ décroît du nombre de comparaisons effectuées.

Analyse de complexité (3)

- Avant les fusions : $\Phi(n_1, \ldots, n_B) = \sum_{j \leq B} n_j \log_{\frac{3}{2}}(\frac{n}{n_j})$.

$$\begin{aligned}
\sum_{j \leq B} n_j \log_{\frac{3}{2}}(\frac{n}{n_j}) &= n \left(\sum_{j \leq B} \frac{n_j}{n} \log_{\frac{3}{2}}(\frac{n}{n_j}) \right) \\
&\leq n \log_{\frac{3}{2}} \left(\sum_{j \leq B} \frac{n_j}{n} \frac{n}{n_j} \right) \\
&= n \log_{\frac{3}{2}}(B)
\end{aligned}$$

- Après les fusions $\Phi(n) = 0$.

D'où un nombre de comparaisons lors des fusions avec une liste fusionnée inférieur ou égal à $n \log_{\frac{3}{2}}(B)$.

Il y a au plus n comparaisons lors des fusions entre listes initiales.

D'où un nombre total de comparaisons en $O(n(\log(B) + 1))$!

File de priorité

Une file de priorité est un ensemble de paires (clé,valeur)
dans un espace de valeurs ordonné.

Les opérations usuelles sont :

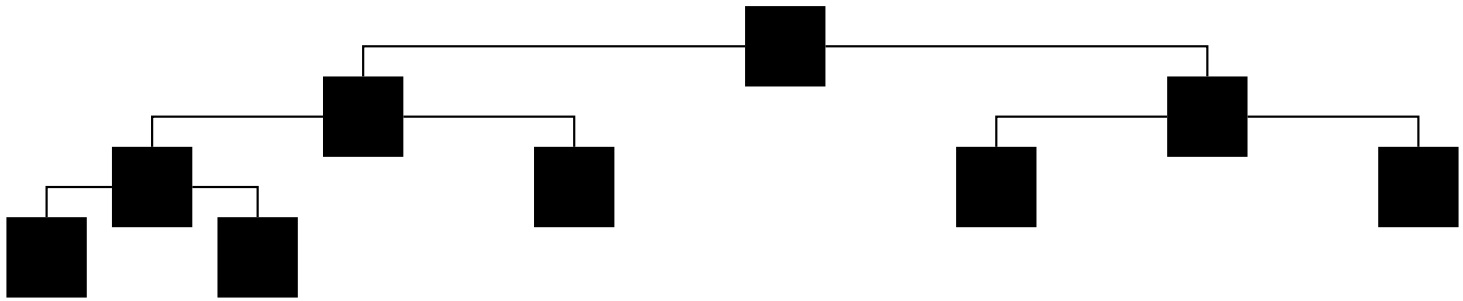
- FileVide() qui teste si la file est vide ;
- Insérer(Id,val) qui insère une nouvelle clé et sa valeur ;
- Minimum() qui renvoie la valeur minimale (et sa clé) de la file (non vide) ;
- ExtraireMin() qui extrait la valeur minimale (et sa clé) de la file (non vide) et l'extrait de la file ;
- DiminuerCle(Id,val) remplace la valeur référencée par Id ;
- Supprimer(Id) remplace la valeur référencée par Id ;

Dans la suite, on ne représentera que les valeurs.

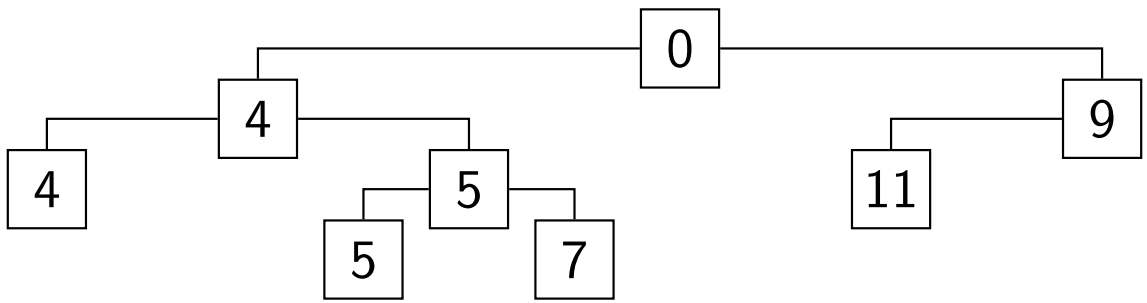
Tas binaire

Un arbre binaire de n sommets est *parfait* si :

- La hauteur de l'arbre h est égale à $\lfloor \log_2(n) \rfloor$;
- Il y a $2^{h'}$ sommets pour tout $h' < h$;
- Les $n - 2^h + 1$ feuilles de hauteur h sont regroupées à gauche.



Un arbre de valeurs est un *arbre tournoi* si la valeur d'un sommet est inférieure ou égale à celles de ses fils.

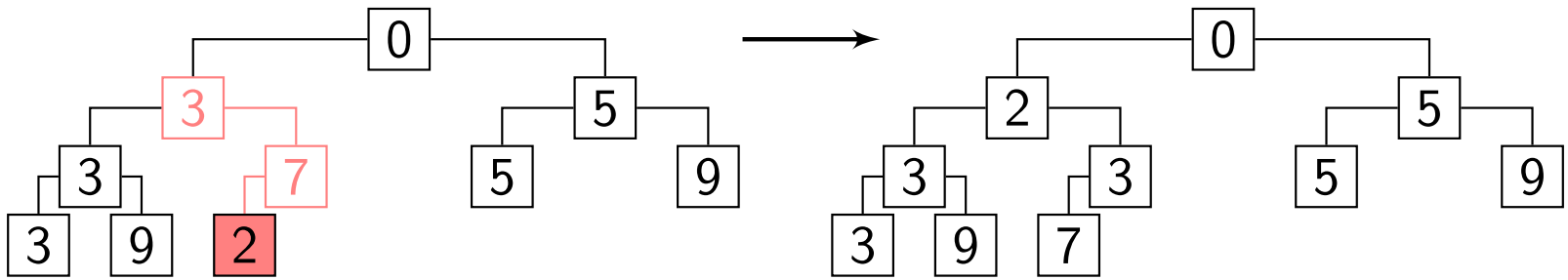


Un *tas binaire* est un arbre tournoi parfait.

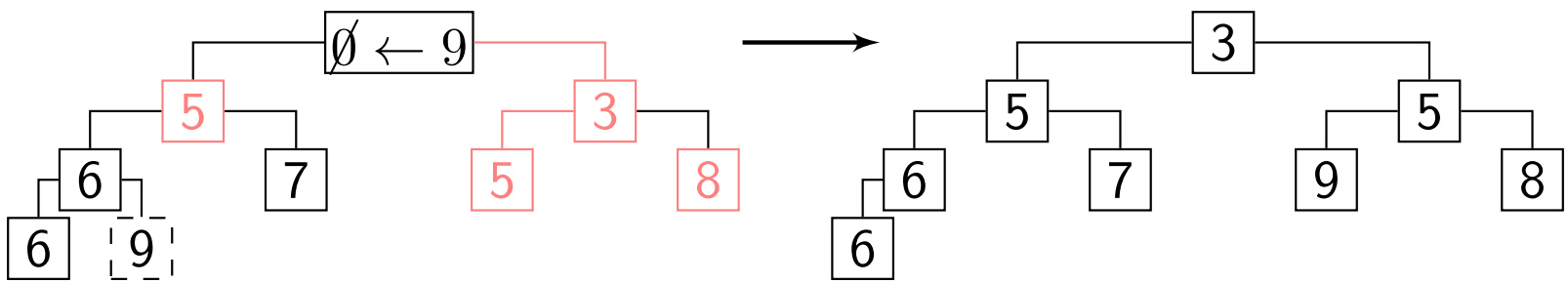
Opérations d'un tas binaire

Le minimum est renvoyé en $O(1)$.

L'insertion s'effectue en $O(\log(n))$.



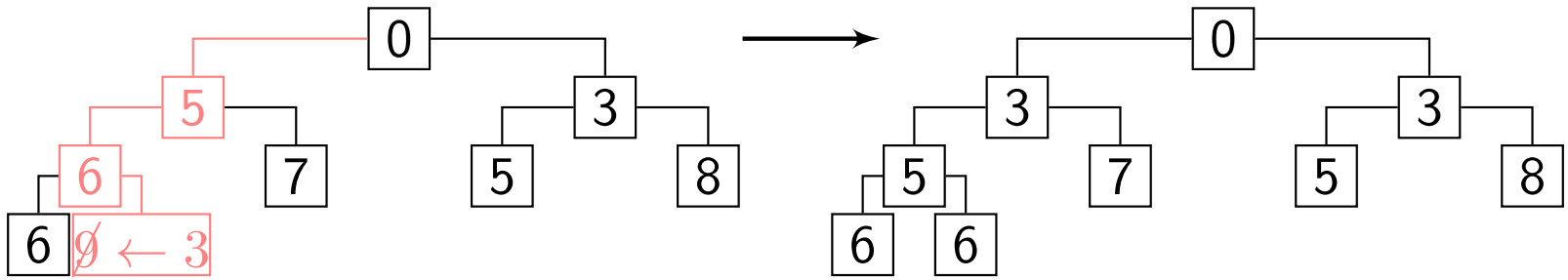
L'extraction du minimum s'effectue en $O(\log(n))$.



Opérations auxiliaires

Diminuer valeur

- Change la valeur si la nouvelle valeur est inférieure ;
- Remonte la valeur vers la racine ;



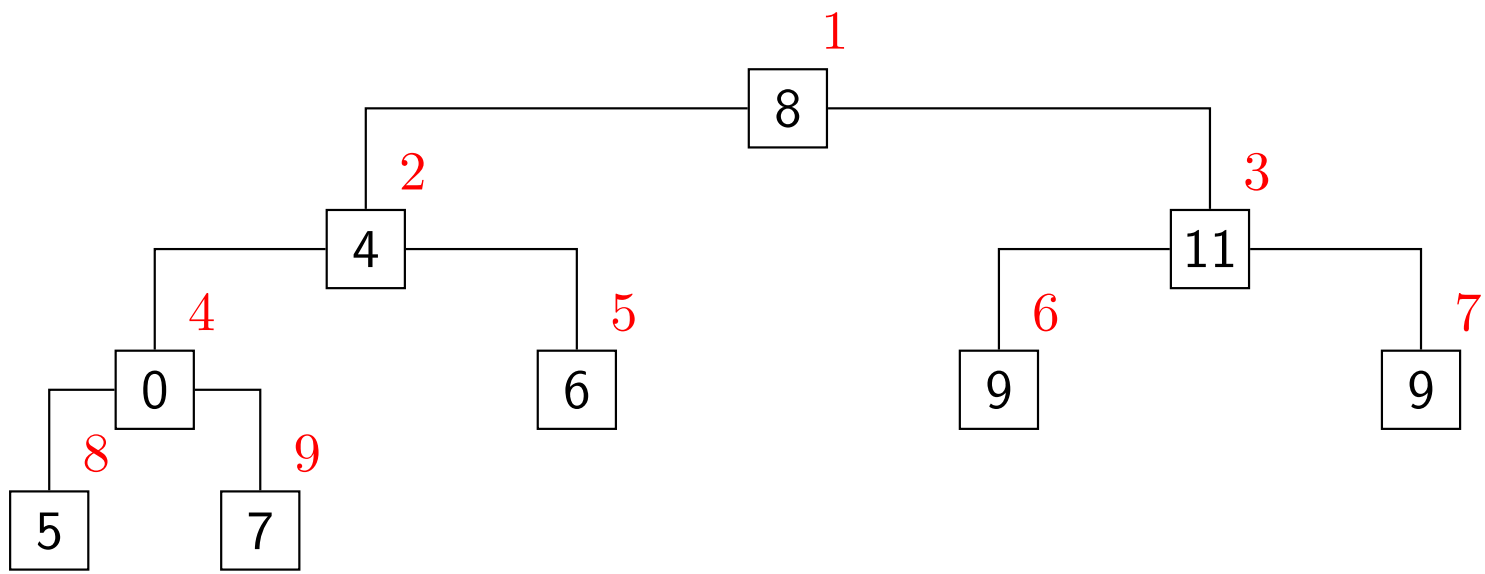
Supprimer valeur

- Affecte la valeur $-\infty$;
- Extrait la valeur minimale $(-\infty)$;

Tas binaire et tableau

Soit T un tableau de n valeurs à organiser en tas binaire.

- La racine est $T[1]$;
- Le fils gauche de $T[i]$ est $T[2i]$ (pour $2i \leq n$) ;
- Le fils droit est $T[2i+1]$ (pour $2i < n$).



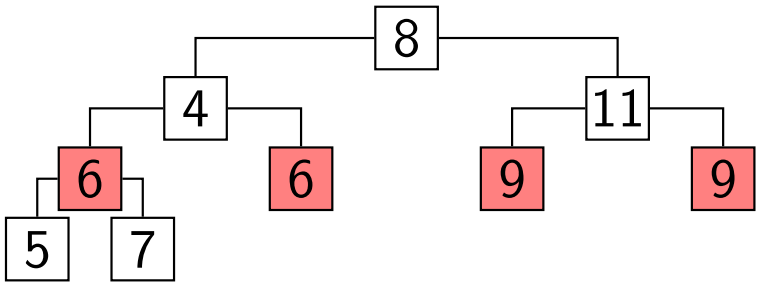
Construction du tas binaire

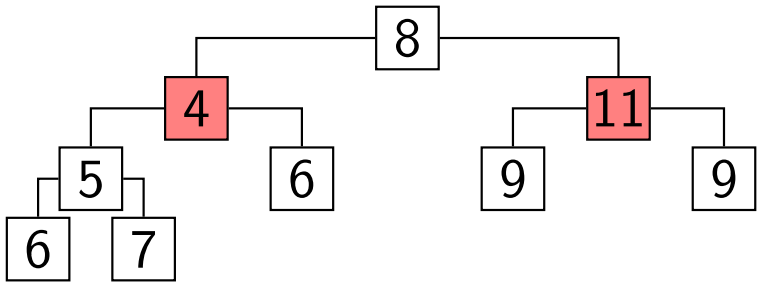
On construit le tas binaire par hauteur décroissante et de gauche à droite :

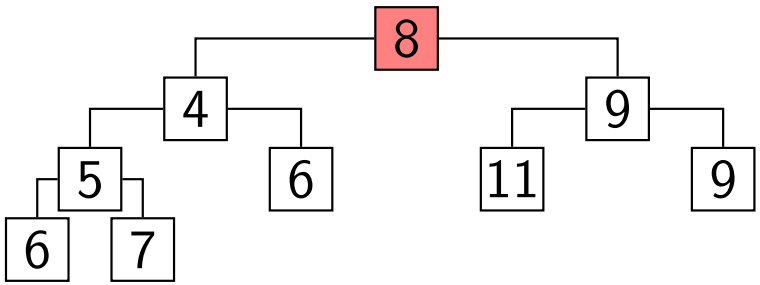
- La valeur du sommet courant est échangée avec la plus petite valeur de ses fils si cette valeur est inférieure.
- On itère ce procédé jusqu'aux feuilles si nécessaire.

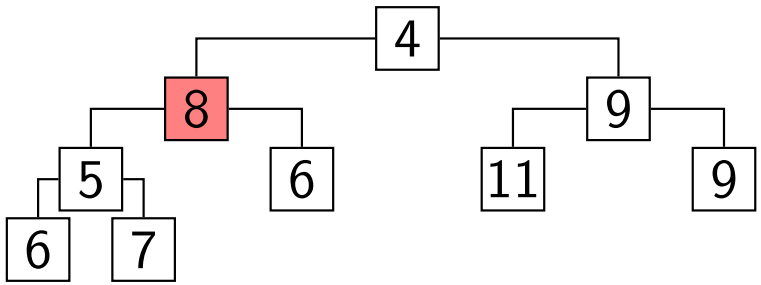
$i \leftarrow 2^{\lfloor \log_2(n) \rfloor - 1}$;
While $i \geq 1$ **do**
  For ℓ **from** i **to** $2i - 1$ **do**
    $j \leftarrow \ell$;
    Repeat
      $done \leftarrow$ **true** ; $k \leftarrow 2j$;
      If $2j < n$ **and** $T[2j] > T[2j+1]$ **then** $k \leftarrow 2j + 1$;
      If $k \leq n$ **and** $T[k] > T[j]$ **then swap**(T, j, k) ; $j \leftarrow k$; $done \leftarrow$ **false** ;
    Until $done$;
  $i \leftarrow i \div 2$;

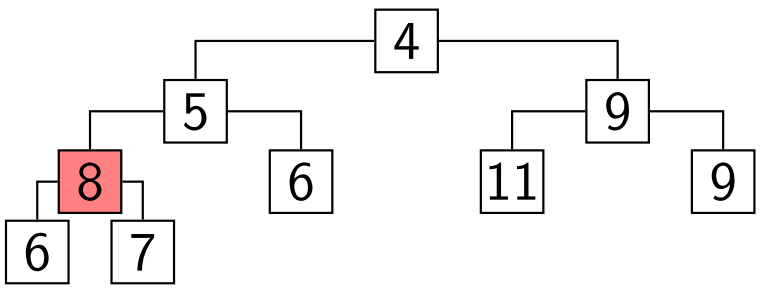
Illustration

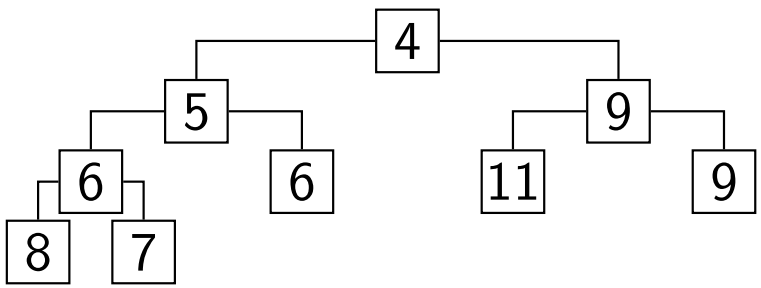












Analyse de complexité

La construction du tas binaire s'effectue en temps linéaire

Preuve.

Chaque sommet de hauteur h' donne lieu à un temps de calcul borné par $c(h-h')$.
D'où un temps de calcul borné par :

$$
\begin{aligned}
c \sum_{h' \leq h-1} 2^{h'}(h-h') = & \quad c \sum_{h' \leq h-1} \sum_{h'' \leq h'} 2^{h''} \\
\leq & \quad c \sum_{h' \leq h-1} 2^{h'+1} \\
\leq & \quad c 2^{h+1} \\
\leq & \quad 2cn
\end{aligned}
$$

Tri par tas

Le tri par tas du tableau T procède ainsi :

- On construit le tas (max) binaire de T en $O(n)$;
- On extrait consécutivement les $n-1$ maximum en les plaçant en $T[n], T[n-1], \ldots$ à la place libérée par les cellules ;
- Cette opération s'effectue en $O(n \log(n))$.

Le tri par tas est donc optimal en temps et en espace.

Plan

Tris par comparaison

2 Borne inférieure de complexité

Tris sans comparaison

Applications

Bornes inférieures de complexité

Soit tp un type de données qui permet uniquement :

- les affectations ;
- les comparaisons.

On considère un algorithme $\mathcal{A}$ qui trie un tableau T de n données de type tp.
On suppose que T contient initialement une permutation des données $v_1 < v_2 < \ldots < v_n$ où les v_i sont toutes différentes des constantes de $\mathcal{A}$.

Soit σ une permutation de $\{1, \ldots, n\}$ et s un état de $\mathcal{A}$.
L'état $\sigma(s)$ est obtenu en :

- en conservant les mêmes informations de contrôle (appels en cours, prochaines instructions, etc.) ;
- en conservant tout donnée $v \notin \{v_1, v_2 \ldots, v_n\}$;
- en substituant à toute donnée v_i la donnée $v_{\sigma(i)}$.

Evolution des états

Observation critique. Considérons les états s et $\sigma(s)$ et op la prochaine instruction à exécuter.

Supposons que $s \xrightarrow{\text{op}} s'$. Si :

- op n'est pas une comparaison de données de tp
- ou op est une comparaison de données de tp
  et renvoie le même résultat pour s et $\sigma(s)$

Alors $\sigma(s) \xrightarrow{\text{op}} \sigma(s')$.

Dans le cas d'une comparaison sur tp qui ne renvoie pas le même résultat, on parle de comparaison *discriminante*.

Une propriété des arbres binaires

La hauteur moyenne des feuilles d'un arbre binaire comportant n feuilles est supérieure ou égale à $\log_2(n)$.

Preuve par récurrence. *(cas $n = 1$ trivial)*

Soit un arbre binaire $\mathcal{A}$ comportant n feuilles.

Soient $\mathcal{A}_g$ et $\mathcal{A}_d$ ses sous-arbres comportant p et $n-p$ feuilles.

Par induction, la somme des hauteurs des feuilles de $\mathcal{A}_g$ (resp. $\mathcal{A}_d$) est supérieure ou égale à $p\log_2(p)$ (resp. $(n-p)\log_2(n-p)$).

Par conséquent, la somme des hauteurs des feuilles de $\mathcal{A}$ est supérieure ou égale à $n + p\log_2(p) + (n-p)\log_2(n-p)$.

La fonction $x\log_2(x)$ est convexe : $\frac{1}{2}(p\log_2(p) + (n-p)\log_2(n-p)) \geq \frac{n}{2}\log_2(\frac{n}{2})$.

D'où : $n + p\log_2(p) + (n-p)\log_2(n-p) \geq n + n(\log_2(n) - 1) = n\log_2(n)$.

□

Un arbre binaire

On construit itérativement un arbre binaire de la façon suivante :

- La racine r contient les $n!$ états initiaux de $\mathcal{A}$ correspondant aux permutations de $\{v_1, v_2 \ldots, v_n\}$.

  Pour toute paire d'états $s_0 \neq s'_0$
  il existe une permutation $\sigma \neq$ id telle que $s'_0 = \sigma(s_0)$.

- Etant donné un sommet u de l'arbre contenant un sous-ensemble S d'états identiques à une permutation près, il y a deux possibilités :
  - soit l'exécution à partir de ces états se termine sans rencontrer une comparaison discriminante.

    Alors u est une feuille de l'arbre.
  - soit l'exécution à partir de ces états rencontre une comparaison discriminante. Alors u a deux fils contenant $S_\top$ et $S_\bot$ les états atteints respectivement après un résultat positif ou négatif de la comparaison.

Analyse de l'arbre binaire

Toutes les feuilles de l'arbre contiennent des singletons.

Preuve.

Considérons une feuille de l'arbre.

Supposons que le sous-ensemble d'états associé n'est pas un singleton.

Considérons s et $\sigma(s)$ avec $\sigma \neq \text{id}$, deux états atteints après terminaison de $\mathcal{A}$.

Alors T n'est pas trié dans l'un des deux états.

□

Soit une distribution uniforme sur les permutations de $\{1, \ldots, n\}$.
Alors le nombre moyen de comparaisons discriminantes de $\mathcal{A}$
est supérieur ou égal à $\log_2(n!) = \Theta(n \log(n))$.

Preuve.

Il y a $n!$ feuilles dans cet arbre binaire.

□

Plan

Tris par comparaison

Borne inférieure de complexité

3 Tris sans comparaison

Applications

Tri par comptage

Hypothèse supplémentaire : tp $= \{0, 1, \ldots, k\}$.

Principe.

- Compter le nombre d'occurrences de $0 \leq i \leq k$ dans T ;
- Remplir T avec ces occurrences par i croissant.

TriComptage(T, n, k)

For i **from** 1 **to** k **do** $Cpt[i] \leftarrow 0$

For i **from** 1 **to** n **do** $Cpt[T[i]] \leftarrow Cpt[T[i]] + 1$

$\ell \leftarrow 1$;

For i **from** 1 **to** k **do**

  For j **from** 1 **to** $Cpt[i]$ **do** $T[\ell] \leftarrow i$; $\ell \leftarrow \ell + 1$

Complexité en $\Theta(n + k)$

Tri partiel par dénombrement

Σ est un alphabet ordonné. Σ^k est ordonné lexicographiquement.

T est un tableau de clés appartenant à Σ^k.

On veut trier partiellement T selon la valeur de la position $m \leq k$

TriP(T, m)

For $a \in \Sigma$ **do** $nb[a] \leftarrow 0$;

For i **from** 1 **to** n **do** $nb[T[i][m]] \leftarrow nb[T[i][m]] + 1$; $T'[T[i][m]][nb[T[i][m]] \leftarrow T[i]$;

$i \leftarrow 1$;

For $a \in \Sigma$ **do**

  For j **from** 1 **to** $nb[a]$ **do** $T[i] \leftarrow T'[a][j]$; $i \leftarrow i + 1$;

Observations.

La complexité est $\Theta(n + |\Sigma|)$.

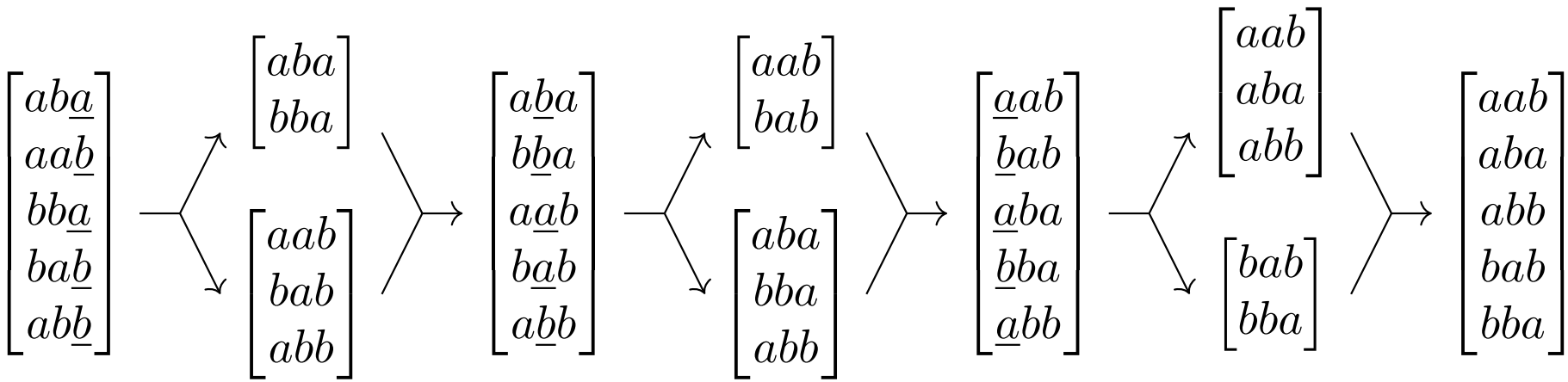
Ce n'est pas un tri sur place.

C'est un tri stable.

Tri par base

On itère le tri par dénombrement de la dernière à la première position :

For m **from** k **downto** 1 **do** TriP(T, m)



Observations.

$$t(n) = \Theta(k(n + |\Sigma|))$$

Si $|\Sigma| = \Theta(n)$ et $k = \Theta(1)$,

on a un algorithme en $\Theta(n)$ pour un espace de clés en $n^{\Theta(1)}$.

Tri des NNI

Le numéro d'identification national est constitué de 13 chiffres.

Il y a approximativement 2^{26} français.

Tri par comparaison

$$n \log(n) \approx 26\ 2^{26}$$

Tri par base

$$k(n + |\Sigma|) \approx 13\ 2^{26}$$

Pas de différence significative.

Plan

Tris par comparaison

Borne inférieure de complexité

Tris sans comparaison

4 Applications

Optimisation discrète

Un problème générique.

- Soit E, un ensemble fini ;
- $Adm \subseteq 2^E$, un ensemble de sous-ensembles admissibles tel que le test $E' \in Adm$? se fasse en temps polynomial ;
- r, une fonction « récompense » de E dans $\mathbb{N}^*$.
- On cherche $E' \in Adm$ tel que $r(E') = \sum_{e \in E'} r(e)$ soit maximale.

Quelques exemples.

- Clique maximale dans un graphe ;
- Somme de sous-ensembles d'entiers inférieure à un seuil ;
- Arbre couvrant de poids maximal ;

Certains problèmes sont NP-complets (voir le cours de complexité) et d'autres admettent des algorithmes en temps polynomial.

Famille libre maximale

Soit un ensemble de vecteurs $\{\mathbf{v_i}\}_{i\leq n}$.

On cherche $I \subseteq \{1,\ldots,n\}$ de taille maximale
telle que $\{\mathbf{v_i}\}_{i\in I}$ soit une famille libre.

Un algorithme en temps polynomial.

$I \leftarrow \emptyset$;
For i **from** 1 **to** n **do**
  If $\{\mathbf{v}_j\}_{j\in I} \cup \{\mathbf{v}_i\}$ est une famille libre **then** $I \leftarrow I \cup \{i\}$;
Return(I) ;

Invariant de boucle.

Il existe I^* tel que $\{\mathbf{v}_j\}_{j\in I}$ peut être complétée en une famille libre maximale $\{\mathbf{v}_j\}_{j\in I^*}$ par des vecteurs de $\{\mathbf{v}_i,\ldots,\mathbf{v}_n\}$.

- Si i n'est pas ajouté à I alors $i \notin I^*$.
- Si i est ajouté à I alors $\mathbf{v}_i = \sum_{j\in I^*} \alpha_j \mathbf{v}_j$ avec $\alpha_k \neq 0$ pour un $k \in I^* \setminus I$.
- D'ou $\{\mathbf{v}_j\}_{j\in I^*\setminus\{k\}\cup\{i\}}$ est une famille libre maximale.

Matroïde

(E, Adm) **est un matroïde si :**

- Adm est non vide et clos par inclusion :
  $E' \in Adm \wedge E'' \subseteq E' \Rightarrow E'' \in Adm$;
- Si $E', E'' \in Adm$ et $|E'| < |E''|$,
  il existe $e \in E'' \setminus E'$ tel que $E' \uplus \{e\} \in Adm$.

Observation. Les ensembles admissibles maximaux ont la même taille.

Un algorithme générique.

Trier E par récompense décroissante ;
$F \leftarrow E$; $E' \leftarrow \emptyset$;
Repeat
  $e \leftarrow$ Extraire(F) ;
  If $E' \cup \{e\} \in Adm$ **then** $E' \leftarrow E' \cup \{e\}$;
Until $F = \emptyset$;
Return(E') ;

Preuve de l'algorithme

Invariant de boucle.

> E' peut être complété en un ensemble admissible de récompense maximale E^* par des éléments de F.

Soit le matroïde (F, Adm') défini par $F' \in Adm'$ si $F' \cup E' \in Adm$.

Par hypothèse d'induction,

F^* est un ensemble admissible de récompense maximale de F
ssi $F^* \cup E'$ est admissible de récompense maximale de E.

• $E' \cup \{e\} \notin Adm$. e ne peut appartenir à un ensemble admissible de récompense maximale de F.

• $E' \cup \{e\} \in Adm$. Soit F^*, admissible de récompense maximale de F.

Cas 1 $\{e\} \in F^*$. immédiat.

Cas 2 $\{e\} \notin F^*$. Si $|F^*| > 1$, il existe $f_1 \in F^*$ tel que $\{e, f_1\}$ soit admissible.

En itérant, on obtient $F^+ = \{e, f_1, \ldots, f_k\}$ admissible tel que :

$$F^* = \{f_0, \ldots, f_k\}$$

Puisque $r(e) \geq r(f_0)$, F^+ a une récompense maximale dans F

et donc $E' \cup F^+$ a une récompense maximale dans E.

Arbre couvrant de poids maximal

Soit un graphe connexe $G = (V, E)$ dont les arêtes sont pondérées.
On cherche un arbre couvrant de poids maximal.

Le matroïde est l'ensemble des arêtes
dont les sous-ensembles admissibles forment des graphes acycliques.

Preuve

Soit E_1, E_2 des ensembles formant des graphes acycliques avec $|E_1| < |E_2|$.
Il existe une décomposition $G_1 = \biguplus_{j \leq m} T_{1,j}$ et $G_2 = \biguplus_{j \leq n} T_{2,j}$
telle que $T_{i,j} = (V_{i,j}, E_{i,j})$ soit un arbre et $E_i = \biguplus E_{i,j}$.

Cas 1 $\exists \{u, v\} \in E_2 \wedge \{u, v\} \cap \biguplus_j V_{1,j} = \emptyset$. Alors $E_1 \cup \{\{u, v\}\}$ convient.

Cas 2 $\exists j \ \exists \{u, v\} \in E_2 \wedge u \in V_{1,j} \wedge v \notin V_{1,j}$. Alors $E_1 \cup \{\{u, v\}\}$ convient.

Cas 3 $\forall \{u, v\} \in E_2 \ \exists j \ \{u, v\} \subseteq V_{1,j}$.
Alors $\forall j' \leq n \ \exists j \leq m \ V_{2,j'} \subseteq V_{1,j}$ ce qui implique $|E_2| \leq |E_1|$. (pourquoi ?)

Observation

L'algorithme de Kruskal est une adaptation à l'arbre couvrant de poids minimal.

Ordonnancement de tâches

Soit un ensemble de tâches I, *de même durée*
avec date d'échéance d_i et pénalité r_i si non respect.
On cherche un ordonnancement qui minimise les pénalités.

Observation

Ceci équivaut à maximiser les pénalités des tâches qui respectent leur échéance.

Le matroïde est l'ensemble des tâches
dont les sous-ensembles admissibles sont ordonnançables sans pénalité.

Preuve

Soit $J = i_1 \dots i_m$ et $J' = j_1 \dots j_n$ des sous-ensembles « ordonnancés »
avec $m < n$. Soit j_k la dernière tâche de J' qui n'appartient pas à J.

On ordonnance $J \cup \{j_k\}$ comme suit :

- On ordonnance d'abord $J \setminus \{j_{k+1}, \dots, j_n\}$ selon l'ordre de J ;
- Puis on place $\{j_k, \dots, j_n\}$.

Toutes les tâches de cet ordonnancement respectent leur échéance. (pourquoi ?)

Comment tester efficacement si un sous-ensemble est ordonnançable ?

Généralisation : algorithmes gloutons

Un problème générique : Un espace de solutions

- structuré par une relation d'extension ;
- contenant une solution minimale ;
- telle que la récompense des solutions soit croissante par extension.

Un algorithme glouton générique.

$sol \leftarrow solmin$;
While Extend$(sol) \neq \emptyset$ **do**
  $rew \leftarrow -\infty$;
  For $sol' \in$ Extend(sol) **do**
    If $r(sol') > rew$ **then** $rew \leftarrow r(sol')$; $sol \leftarrow sol'$;
Return(sol) ;

Selon les problèmes, les algorithmes gloutons fournissent :

- la solution optimale ;
- une solution approchée avec garantie d'approximation.

Plus proche paire de points

Problème. Déterminer une paire de points du plan la plus proche parmi un ensemble de n points du plan.

Observation. L'algorithme naïf a une complexité $\Theta(n^2)$.

Etape préliminaire. (en $\Theta(n\log(n))$)
Créer le tableau X d'indices de T trié selon le couple (abcisse,ordonnée).
Créer le tableau Y d'indices de T trié selon les ordonnées.

Illustration.

$$T = \begin{bmatrix}(3.5, 0.7)\\(2.5, 0.1)\\(2.5, 0.9)\\(4, 0.2)\\(2, 1.5)\end{bmatrix} \qquad X = \begin{bmatrix}5\\2\\3\\1\\4\end{bmatrix} \qquad Y = \begin{bmatrix}2\\4\\1\\3\\5\end{bmatrix}$$

Diviser pour régner (1)

Partitionner selon les coordonnées les points en deux sous-ensembles de tailles approximativement égales.

Part(X, Y, n)

Partionnement de X trivial

$mid \leftarrow \lfloor \frac{n}{2} \rfloor$; $xmid \leftarrow T[X[mid]].x$; $ymid \leftarrow T[X[mid]].y$;

For i **from** 1 **to** mid **do** $X_g[i] \leftarrow X[i]$;

For i **from** $mid + 1$ **to** n **do** $X_d[i - mid] \leftarrow X[i]$;

Partionnement de Y à l'aide des coordonnées

$i_g \leftarrow 1$; $i_d \leftarrow 1$

For i **from** 1 **to** n **do**

  If $T[Y[i]].x < xmid$ **or** $(T[Y[i]].x = xmid$ **and** $T[Y[i]].y \leq ymid)$ **then**

    $Y_g[i_g] \leftarrow Y[i]$; $i_g \leftarrow i_g + 1$;

  Else

    $Y_d[i_d] \leftarrow Y[i]$; $i_d \leftarrow i_d + 1$;

return(X_g, Y_g, X_d, Y_d)

Diviser pour régner (2)

Résoudre les sous-problèmes.

Short(X, Y, n)

Cas de base $n \leq 2$

If $n = 1$ **then return** ∞

If $n = 2$ **then return**$(\sqrt{(T[X[2]].x - T[X[1]].x)^2 + (T[X[2]].y - T[X[1]].y)^2})$;

d minimum des deux plus courtes distances

$(X_g, Y_g, X_d, Y_d) \leftarrow$ Part(X, Y, n) ;

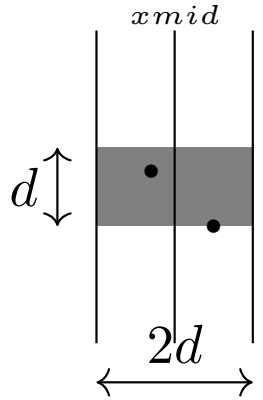
$d \leftarrow$ Short$(X_d, Y_d, \lceil \frac{n}{2} \rceil)$;

If $n \geq 4$ **then** $d \leftarrow \min(d,$ Short$(X_g, Y_g, \lfloor \frac{n}{2} \rfloor))$;

Prendre en compte les distances entre points gauches et droits.

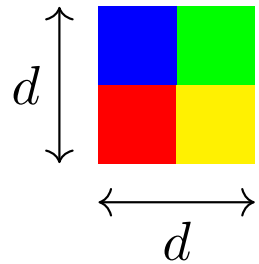
...

Si la distance entre un point *gauche* et un point *droit* est inférieure à d, alors :



Diviser pour régner (3)

Observation. Soit un ensemble de points de distance minimale d,
il y a au plus 4 points dans un carré $d \times d$.
Preuve. Au plus un point par carré de couleur.



Pour diminuer la distance minimale :

- Sélectionner dans Y les points dans la bande $xmid - d \leq x \leq xmid + d$.
- Parcourir les points de la sélection Y' (selon les ordonnées).
- Calculer la distance d'un point à ses 7 successeurs.

Diviser pour régner (4)

Implémentation.

$xmid \leftarrow T[X[\lfloor \frac{n}{2} \rfloor]].x$;

Sélection

$i' \leftarrow 0$;

For i **from** 1 **to** n **do**

If $T[Y[i]].x \geq xmid - d$ **and** $T[Y[i]].x \leq xmid + d$ **then**

$i' \leftarrow i' + 1$; $Y'[i'] \leftarrow Y[i]$;

Calcul des distances

For i **from** 1 **to** $i' - 1$ **do**

For j **from** $i + 1$ **to** $\min(i + 7, i')$ **do**

$d' \leftarrow \sqrt{(T[Y'[j]].x - T[Y'[i]].x)^2 + (T[Y'[j]].y - T[Y'[i]].y)^2}$;

If $d' < d$ **then** $d \leftarrow d'$;

return(d) ;

$$t(n) = 2t(\frac{n}{2}) + \Theta(n) \Rightarrow t(n) = \Theta(n \log(n))$$

Tests géométriques

Soient $p_1, p_2, p_3 \in \mathbb{R}^2$ des points non alignés et p un point quelconque.

- p appartient au triangle formé par p_1, p_2 et p_3
  ssi l'unique solution $(\lambda_1, \lambda_2, \lambda_3)$ de :

$$\sum_{i \leq 3} \lambda_i p_i = p \qquad \wedge \qquad \sum_{i \leq 3} \lambda_i = 1$$

  appartient à $(\mathbb{R}^+)^3$.

- p appartient au cone d'origine p_1, et de directions $\overrightarrow{p_1 p_2}$ et $\overrightarrow{p_1 p_3}$
  ssi l'unique solution (λ_2, λ_3) de :

$$\lambda_2 \overrightarrow{p_1 p_2} + \lambda_3 \overrightarrow{p_1 p_3} = \overrightarrow{p_1 p}$$

  appartient à $(\mathbb{R}^+)^2$.

Ces tests s'opèrent en temps constant.

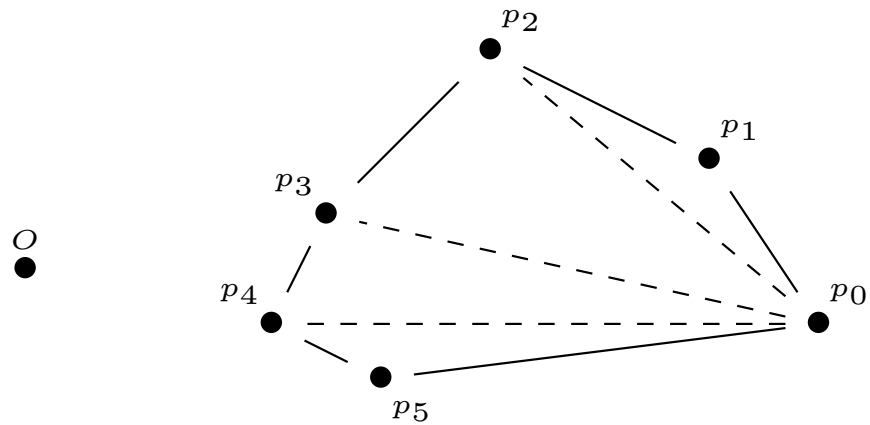
Polygone convexe

Un polygone $\mathcal{P}$ est donné par une liste $p_0, \ldots, p_{n-1}$ de ses sommets tels que la liste $(\overline{p_i p_{i+1\%n}})_{0 \leq i < n}$ soit celle de ses segments parcourue en sens direct.

Il est convexe ssi tous les angles vus de l'intérieur sont saillants.

Test (naïf) d'appartenance en $O(n)$

$$O \in \mathcal{P} \text{ ssi } \exists 0 < i < n-1 \; O \in Triangle(p_0, p_i, p_{i+1\%n})$$



Ajout d'un sommet

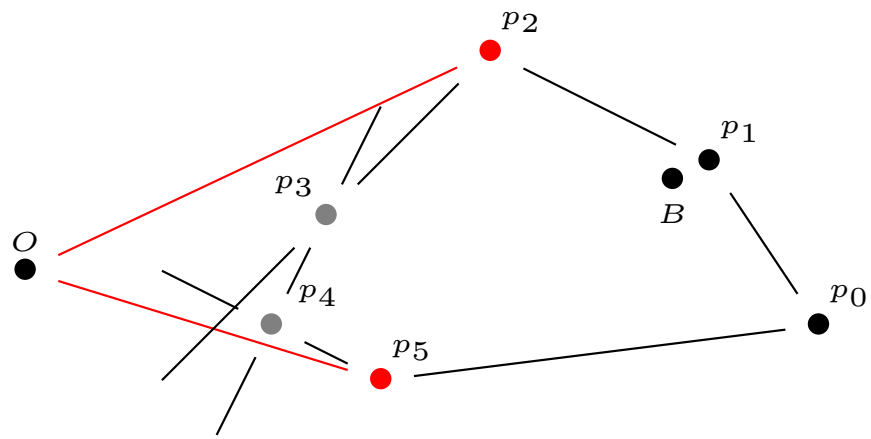
Soit $O \notin \mathcal{P}$, polygone convexe.

- Calcul de $B = \frac{1}{3}(p_0 + p_1 + p_2)$;
- Pour tout $i < n$, extraction de p_i ssi :

$$O \in Cone(p_i, \overrightarrow{p_{i-1\%n}p_i}, \overrightarrow{p_{i+1\%n}p_i})$$

- Parmi les n' points restants, insertion de O entre $p_{\alpha(i)}$ et $p_{\alpha(i+1\%n')}$ tels que :

$$O \in Cone(B, \overrightarrow{Bp_{\alpha(i)}}, \overrightarrow{Bp_{\alpha(i+1\%n')}})$$



Enveloppe convexe

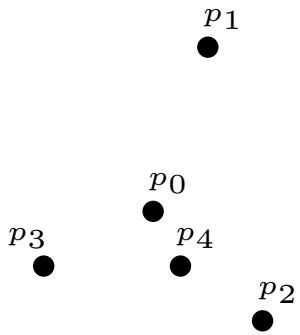
Soit S un ensemble de $n \geq 3$ points $p_0, \ldots, p_{n-1}$ dans le plan.

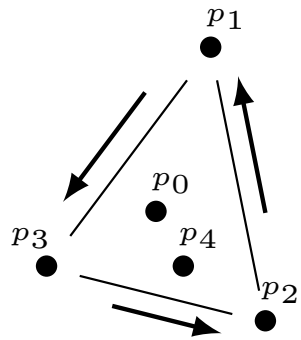
$EC(S)$, l'enveloppe convexe de S, est définie par :

$$EC(S) = \{p \mid \exists \{\lambda_i\}_{i<n} \in (\mathbb{R}^+)^n \wedge \sum_{i<n} \lambda_i = 1 \wedge p = \sum_{i<n} \lambda_i p_i\}$$

$EC(S)$ est un polygone (convexe)

dont l'ensemble des sommets S' est un sous-ensemble de S.





Objectif. Construire une représentation de $EC(S)$.

- Algorithme de Graham
- Algorithme dichotomique
- Algorithme de Jarvis
- Algorithme de Chan

Intérieur de l'enveloppe convexe

$\overset{\circ}{EC}(S)$, l'intérieur de $EC(S)$, est défini par :

$$\overset{\circ}{EC}(S) = \{p \mid \exists r > 0 \; Disc(p, r) \subseteq EC(S)\}$$

$\overset{\circ}{EC}(S)$ est non vide si et seulement si il existe p_i, p_j, p_k non alignés.

Comment trouver un point de $\overset{\circ}{EC}(S)$?

$i \leftarrow 2$;
While $i < n$ **and** $p_1 = p_i$ **do** $i \leftarrow i + 1$;
If $i \geq n$ **then return**$(\mathbf{false})$;
$j \leftarrow i + 1$;
While $j \leq n$ **and** $p_j \in d(p_1, p_i)$ **do** $j \leftarrow j + 1$;
If $j > n$ **then return**$(\mathbf{false})$;
return$(\frac{1}{3}(p_1 + p_i + p_j))$;

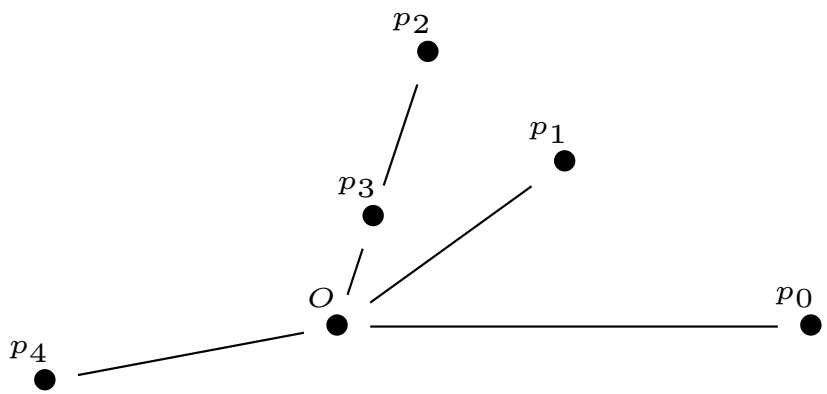
Circuit polaire

Soit deux vecteurs non nuls en coordonnées polaire $\mathbf{v} = (\rho, \theta)$ et $\mathbf{v}' = (\rho', \theta')$.

$$\mathbf{v} \prec \mathbf{v}' \text{ ssi } \theta < \theta' \text{ ou } (\theta = \theta' \text{ et } \rho > \rho')$$

Soit O une *origine* et $S = \{p_0, \ldots, p_{n-1}\}$ un ensemble de points différents de O.

Soit α une permutation. $(p_{\alpha(0)}, \ldots, p_{\alpha(n-1)})$ est un circuit polaire relatif à O si pour tout $i < n$, $\overrightarrow{Op_{\alpha(i)}} \prec \overrightarrow{Op_{\alpha(i+1)}}$ avec $\overrightarrow{Op_{\alpha(0)}}$ comme axe $\overrightarrow{Ox}$.



Le circuit polaire s'obtient par un tri en $O(n \log(n))$.

Eviter les coordonnées polaires

Soit O le point origine et $\overrightarrow{Op_0}$ l'orientation de l'axe des abcisses.

Prec(p, p')

(vecteurs de même direction)

If $\exists\lambda\ \overrightarrow{Op} = \lambda\overrightarrow{Op'} \wedge \lambda > 1$ **then return true**

If $\exists\lambda\ \overrightarrow{Op} = \lambda\overrightarrow{Op'} \wedge 0 < \lambda \leq 1$ **then return false**

(un vecteur de direction $\overrightarrow{Op_0}$)

If $\exists\lambda\ \overrightarrow{Op} = \lambda\overrightarrow{Op_0} \wedge \lambda > 0$ **then return true**

If $\exists\lambda\ \overrightarrow{Op'} = \lambda\overrightarrow{Op_0} \wedge \lambda > 0$ **then return false**

($\overrightarrow{Op}$ appartient au demi-plan supérieur)

If $det(\overrightarrow{Op_0}, \overrightarrow{Op}) > 0$ **then return** $p' \notin Cone(\overrightarrow{Op_0}, \overrightarrow{Op})$

($\overrightarrow{Op}$ de direction opposée à $\overrightarrow{Op_0}$)

If $det(\overrightarrow{Op_0}, \overrightarrow{Op}) = 0$ **then return** $det(\overrightarrow{Op_0}, \overrightarrow{Op'}) < 0$

($\overrightarrow{Op}$ appartient au demi-plan inférieur)

If $det(\overrightarrow{Op_0}, \overrightarrow{Op}) < 0$ **then return** $p' \in Cone(\overrightarrow{Op_0}, \overrightarrow{Op})$

Du circuit polaire à l'enveloppe convexe

Soit $\mathcal{C}$ un circuit polaire de n points (liste circulaire) avec $O \in \overset{\circ}{EC}(\mathcal{C})$.

Choisir $p_0 \in \mathcal{C}$ extrémal pour un ordre lexicographique des coordonnées.

$p \leftarrow p_0$;

While $succ(\mathcal{C}, p) \neq p_0$ **do**

  If $succ(\mathcal{C}, p) \in Triangle(O, p, succ(\mathcal{C}, succ(\mathcal{C}, p))$ **then**

    $Extraire(\mathcal{C}, succ(p))$; **If** $p \neq p_0$ **then** $p \leftarrow pred(\mathcal{C}, p)$;

  Else

    $p \leftarrow succ(\mathcal{C}, p)$;

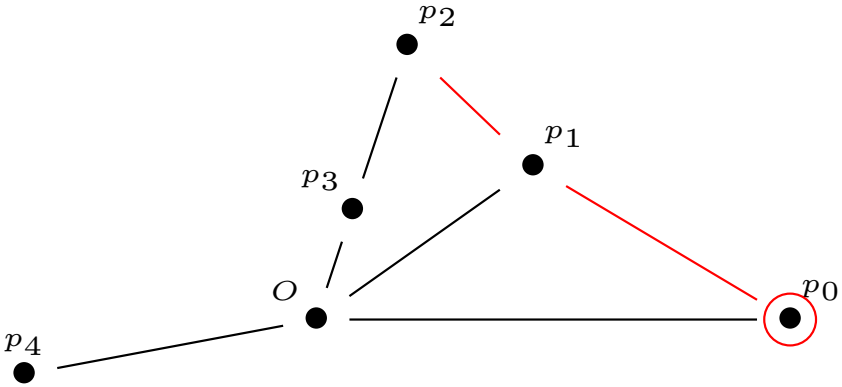
Complexité.

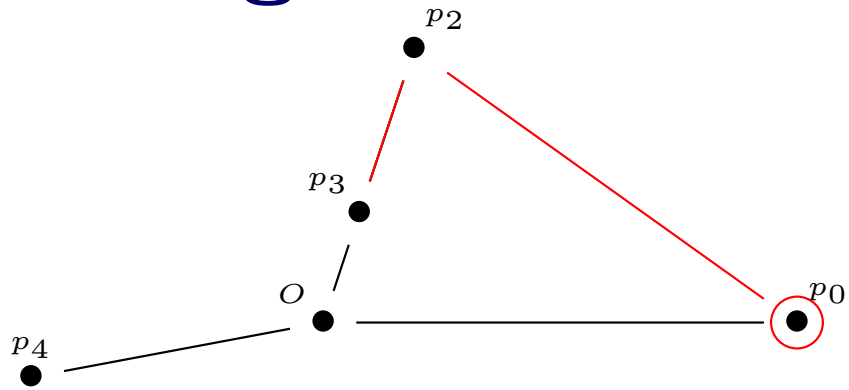
Soit $\ell = |\mathcal{C}| + \min(i > 0 \mid succ^{(i)}(p) = p_0)$.

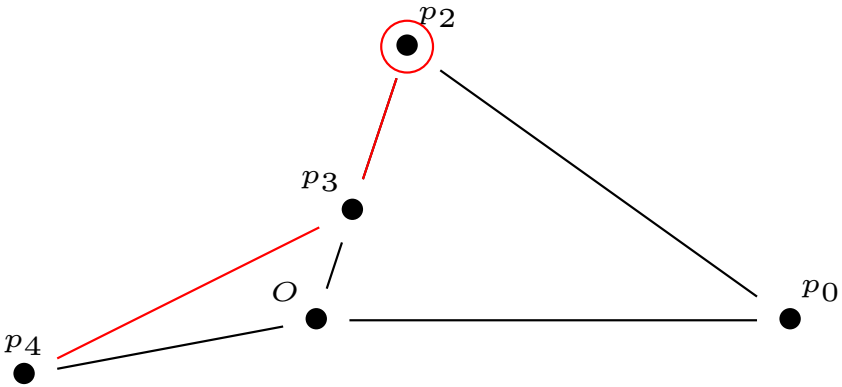
ℓ est initialement égal à $2n$ et décroît à chaque étape.

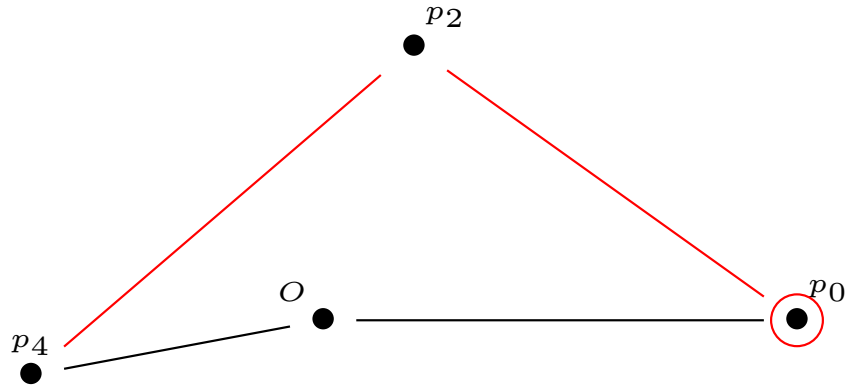
D'où un algorithme en $O(n)$.

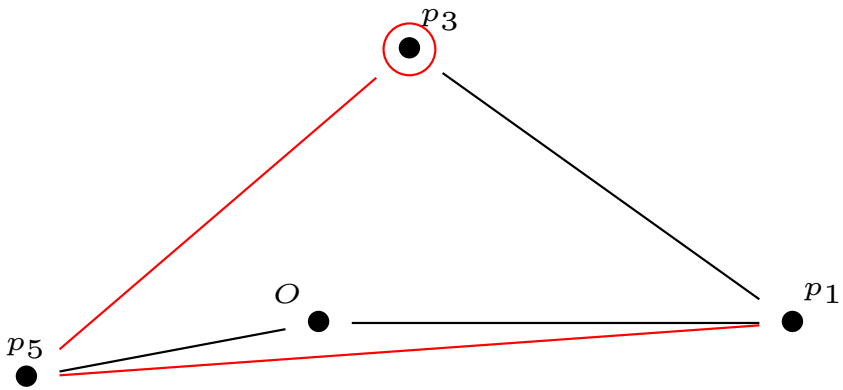
Déroulement de l'algorithme











Correction de l'algorithme

Invariants.

- L'enveloppe convexe de $EC(\mathcal{C})$ reste inchangée.
- Les angles formés par les segments de p_0 à $succ(\mathcal{C}, p)$ sont saillants (vus de O).

Extraction d'un sommet.

- Un sommet retiré appartient à l'intérieur de l'enveloppe convexe ou à l'intérieur d'une arête.
- Puisque p devient $pred(\mathcal{C}, p)$, la deuxième propriété reste vérifiée.

Passage au successeur.

- $\mathcal{C}$ est inchangé.
- La deuxième propriété reste vérifiée par construction.

Observation.

A la fin de l'algorithme l'angle en p_0 est nécessairement saillant (choix de p_0).

Un polygone (simple) dont tous les angles sont saillants est convexe.